Trond Albinussen

NAVIGERING I INTERNETT OG HYPERMEDIA

Hovedfagsavhandling i Informatikk, høsten 2003

Fakultetet for fysikk, informatikk og matematikk

Institutt for Datateknikk og informasjonsvitenskap

"The promise of hypermedia is the ability to produce complex, richly interconnected, and cross-referenced bodies of multimedia information. Unfortunately, hypermedia also has the ability to produce complex, disorganized tangles of haphazardly connected documents"

Utting & Yankelovich: Context and Orientation in Hypermedia Networks, 1989.

Forord

Det har vært en glede å kunne være med å utvikle et grafisk navigeringsprogram fra starten av. Det har gitt god teknisk innsikt i forskjellige programmeringsspråk, og datamaskiner generelt. Det har vært svært motiverende å se at programmet faktisk har vært tilgjengelig for studenter i faget MNFIT222 under selve utviklingsprosessen. Dette har gjort at vi aldri har mistet fokus på brukervennlighet og enkelhet.

Det har naturligvis vært en god del frustrasjoner, spesielt ved utviklingen av det grafiske grensesnittet. Likevel føler vi at det har vært en meget lærerik prosess, og at vi kan være stolte av resultatet. I etterkant kan vi se tilbake og si at vi har utviklet hele greia på egen hånd.

Når jeg skulle velge hovedoppgave, la jeg stor vekt på at den skulle være teknisk orientert. Jeg har stor glede av å utvikle programmer, og denne oppgaven har gitt meg muligheten til å lære et stort utvalg av teknikker og metoder.

Jeg vil takke min veileder Arvid Staupe for å gi meg stor frihet i utformingen av oppgaven og med tekniske løsninger. Han har hjulpet med ideer og inspirasjon. I tider hvor den tekniske utviklingen av programmet har krevd mye detaljert oppmerksomhet, har han aldri mistet det store bildet.

Jeg vil takke Mamma og Pappa for støtten de hele tiden har gitt meg under hovedfagsoppgaven. Hvis jeg skulle dedikert denne oppgaven til noen, måtte det bli til dem. Øistein Gjøvik har vært behjelpelig ved utviklingen av de matematiske ligningene. Jeg skylder også takk til forskjellige anonyme deltakere på Java-forumet til Sun Microsystems. De har ofte vært redningen min.

Trond Albinussen
Trondheim, høsten 2003

Sammendrag

Det overordnede temaet i denne hovedfagsavhandlingen er navigering i Internett og hypermedia, med vekt på grafisk navigering til forskjell fra tekstlig navigering.

Oppgaven består av en praktisk og en teoretisk del, med fortrinn på den praktiske delen. Praksisen består i å lage et grafisk navigasjonsprogram som supplement til det allerede eksisterende hypersystemet i MNFIT222. Utviklingen av dette programmet skal skje på bakgrunn av tidligere implementeringer i andre systemer, både på Internett og i andre applikasjoner. Hovedfokus ligger på bruken av grafiske løsninger. I den teoretiske delen har vi lagt vekt på de generelle problemstillinger om navigeringsproblemer i hypermedia og Internett.

Oppgaven er likevel langt på vei en teknisk oppgave, til forskjell fra en litterær oppgave.

I litteraturen blir navigeringsproblemer i hypertekst ofte beskrevet med ordene *desorientering* og *kognitiv overbelastning*. For å motvirke dette kan man benytte mange forskjellige løsninger og teknikker, blant annet grafiske kart. Vår motivasjonen er ønsket om å gi brukerne et slikt kart. Vi har derfor utviklet programmet **Lupen**.

Lupen er implementert i et eksisterende hypersystem **Opsys**. Opsys bygger på en pensumbok i faget MNFIT222 Operativsystemer, og er et læringsmiljø for mellomfagstudenter ved NTNU. Vi ville finne ut hvor langt vi kunne komme, og samtidlig å dokumentere problemstillingene vi møtte underveis. Lupen er et resultat av en avveiing mellom muligheter og praktisk gjennomførbarhet. Likevel vil vi presentere noen nye tjenester og ideer:

- Lupen er ikke tilstandsløs. Den oppdateres ved brukerens klikk i hypersystemet
- Brukernes historie blir lagret utover nettleserens sesjon.
- Mulighet for å avgjøre om Lupen skal være tilstandsløs.
- Man kan forstørre et område på Lupen.

Den tekniske utviklingen av Lupen har vært krevende, og vi har møtt mange problemer. Vi kan nevne de mest grunnleggende av disse:

- Kode for layouter i nettkart er gjerne patentbeskyttet, eller krever en svært god innsikt i matematikk. Dette gjorde at vi måtte utvikle vår egen layout.
- Opsys arkitektur er basert på egendefinerte Perl-løsninger, og dette måtte vi ta hensyn til. På grunn av dette er ikke Lupen direkte teknisk overførbar til andre systemer.
- Opsys har vært svært dårlig dokumentert.

I Kapittel 1 av oppgaven har vi lagt vekt på Internets historie og utviklingen av nettleserne. Dette vil vise hvordan en økende mengde tilgjengelig informasjon har ført til generelle problemer med navigering, både i Internett og ellers. Kapittel 2 vil vise de generelle teoretiske og kognitive problemene i mer detalj. I Kapittel 3 går vi igjennom noen eksempler på grafiske kart, og deres fordeler og ulemper. Vi har studert disse under utviklingen av Lupen, og dette har gitt oss verdifull innsikt i forskjellige metoder og funksjonalitet for å øke nytteverdien av grafiske kart. Vi vil oppsummere denne innsikten i kapittelet. Kapittel 4 vil presentere det eksisterende hypersystemet Opsys. En kort beskrivelse av arkitekturen vil vi også ta med. De påfølgende 2 kapitlene vil ta for seg Lupen. Utviklingen har basert seg på en hurtig rekkefølge av prototyper, og vi vil derfor i Kapittel 5 oppsummere den innsikten vi fikk av den første funksjonelle prototypen, både teknisk og kognitivt. I Kapittel 6 presenterer vi vår endelige løsning på Lupen. Vi vil ta for oss både funksjonalitet og arkitektur. Kapittel 7 vil oppsummere arbeidet, og gi pekere til et eventuelt fremtidig arbeid med grafiske kart.

Vi bruker ordet **forfatter** som en fellesbetegnelse på utviklere og designere. Forfatter er den eller de personene som utvikler innholdet på nettstedene. **Bruker** er den jevne person som benytter systemene, uten å ta del i selve utformingen. **Webmaster** er ansvarlig for tjenerne og den tekniske utviklingen av et hypersystem.

Innholdsfortegnelse

Figurliste

1 Fra bøker til Internett.

1.1 Innledning

Siden skriftspråket ble oppfunnet for årtusener siden, har det gradvis blitt skapt et behov for å finne fram i større mengder med informasjon. For hver mengde informasjon som var mer enn hva et menneske kunne hanskes med, har det vært nødvendig å strukturere og abstrahere sammenhenger mellom forskjellige dokumenter.

Vi vil gi en kort innføring i historien bak denne struktureringen, og hvordan den har vokst fram til å bli dagens Internett.

Vanlig lineær tekst er endimensjonal, det vil si at den leses fra begynnelse til slutt. For å navigere seg i en slik tekst holder det at brukeren vet hva som er begynnelsen på teksten, og leser fra side en til siste side. Internett er multidimensjonalt, og dette gjør at det ikke alltid er like enkelt å vite hvor man er, hvor man har vært, og hvor man kan gå videre.

Hensikten med dette kapitlet er ikke å gå inn i detaljer om navigering i dokumenter og skriftspråk i fortiden. Men heller å vise den økende kompleksiteten og de utfordringer som Informasjonsalderen har gitt oss.

Nettleserne er de verktøy som brukerne benytter til å lete etter informasjon med. Deres historie er verd å ta med seg, fordi det var først når nettleserne ble tilgjengelig for allmennheten i 1993 at informasjonsalderen ble virkelighet for flere enn ekspertene.

1.2 Klassisk strukturering av informasjon

Måten vi mennesker har oppbevart og strukturert informasjon på, har gjennomgått en sakte utvikling. Et av de første skriftspråk var den sumeriske kileskriften cirka 3000 år før Kristus. Leirtavler som inneholdt skriftspråk ble lagret i templer og andre bygninger. Mye av denne informasjonen gikk tapt i branner, kriger og andre ulykker.

Et av de første bibliotekene var biblioteket i Alexandria, som blant annet inneholdt ”de Dødes Bok” [Rad91]. Det inneholdt på et tidspunkt omkring en halv million dokumenter. Cirka år 300 før Kristus oppfant dette biblioteket systemet med eksterne referanser[1].

I tusen år mellom Romerrikets fall var bøker skrevet på pergament den vanligste typen av skrevet informasjon. De ble gjerne kopiert for hånd av munker. Bøker var tradisjonelt ganske korte, cirka 20–30 sider.

I begynnelsen var det kun de rike og geistlige som hadde råd til bøker. Dette var fordi avskriving for hånd var dyrt og tidkrevende. Gutenberg var ikke den første som oppfant trykkpressen, men han var den som gjorde det praktisk med måte med utskiftbare typer. Han produserte sin første bibel på denne måten cirka år 1453. Sakte, men sikkert ble bøker etter hvert mer tilgjengelige for vanlige folk. Indekssider og nummerering av sidene ble vanlige. Da mengden bøker vokste til ingen enkelt person kunne holde oversikt over dem, ble de i den vestlige verden inndelt i klasser som avspeilet de forskjellige studieretningene i datidens universitetsdisipliner: medisin, klassisk, historie og filosofi.

Bibliografier er pekere eller lenker til dokumenter. Enkle bibliografier ble publisert i middelalderen. På 1700-tallet ble det utviklet bibliografier som tok hensyn til forfattere og emner. Utover 1800-tallet var mengden av dokumenter og publikasjoner så stor at oppdatering av bibliografiene ble svært arbeidskrevende. I 1851 rapporterte direktøren av Smithsonian Library i USA at 20 000 publikasjoner ble publisert årlig, og de sekretærene som oppdaterte bibliografien var i fare for å oversvømmes [Rad91].

Melvin Dewey publiserte i 1876 sitt klassifiseringssystem basert på desimaler. Han delte informasjonen ut i fra den klassiske grupperingen som vitenskapen hadde brukt i århundrer. Denne grupperingen ble så videre delt opp i underklasser, med desimaler som notasjon. Varianter av dette systemet ble raskt normen i alle vestlige biblioteker.

[1] ”Alexandria”. Britannica Encyclopedia Online, http://search.eb.com/ebi/article?eu=2945531

Eksplosjonen av dokumenter var så stor utover det 19. århundre at klassifisering av informasjon ble et eget fag på universitetene[2].

1.3 Utviklingen av moderne informasjonssystemer

1.3.1 Vannevar Bush og Memex

Vannevar Bush var vitenskapelig rådgiver til president Roosevelt under den annen verdenskrig. Han var ansvarlig for å mobilisere og organisere amerikansk forskning til krigen mot Tyskland. Han forandret forskningen fra å være et primært privat industrielt felt, til å være et ansvar for myndighetene. Dette muliggjorde store og uhyre ressurskrevende eksperimenter, som for eksempel Manhattan-prosjektet som utviklet den første atombomben. Denne transformasjonen ble senere kalt Pentagon-systemet, eller det militærindustrielle kompleks[3].

Etter Andre Verdenskrig presenterte Vannevar Bush sin artikkel ”**As we may think**”. I den hevder han at forskere finner det stadig vanskeligere å holde tempoet oppe på grunn av den økte informasjonsmengden. De blir overveldet av konklusjoner, rapporter og data. Mengden gjør det urealistisk å lese alt sammen. De finner det også umulig å holde oversikt over all ny informasjon. I ekstreme tilfeller kan viktige vitenskapelige funn gå tapt i informasjonsmengden, da funnene ikke når de som kan videreforedle og bruke dem. Dette skjer samtidlig som kravet om spesialisering øker, og innsatsen som blir gjort for å fremme tverrfaglighet minsker.

Bush trodde at den tekniske utviklingen i fremtiden kunne gjøre det mulig å lage en maskin som kunne inneholde den totale kunnskapen til menneskeheten. Men for å kunne utnytte en slik maskin effektivt måtte man gjøre forandringer i måten man bruker informasjon på. Bush mente at den nåværende indekseringen av informasjon er kunstig: Når data er plassert i et lager, blir de lagret i henhold til en alfabetisk eller numerisk orden, gjerne en variant av systemet til Dewey. De blir funnet ved å traversere nedover underklasser, og de eksisterer bare på en plass (med mindre duplikater blir brukt).

[2] ”Library”, Encyclopedia Britannica Online, http://search.eb.com/eb/article?eu=109618
[3] ”Bush, Vannevar”, Encyclopedia Britannica Online, http://search.eb.com/eb/article?eu=18540

Reglene for å gjenfinne dem er vanskelige, og når man har funnet ett sett med data, må man gjøre hele sekvensen om igjen for å finne ett nytt sett.

Bush sa i sin artikkel at denne måten å lete etter informasjon på, var gammeldags og ineffektiv. I stedet for å basere informasjon på gammeldagse og rigide regler, er det bedre å benytte den mer dynamiske metoden menneskehjernen bruker. Menneskehjernen arbeider ut i fra assosiasjon: Man hopper fra ting til ting i henhold til et ukjent og intrikat sett med regler. Assosiasjoner som blir lite brukt kan forsvinne, data er ikke permanente, og minnet blir hele tiden oppdatert og revidert. Den teknologiske utviklingen kunne en dag gjøre det mulig å implementere et slikt system i en maskin. Han kalte sin teoretiske maskin **Memex**.

Bush så for seg at brukeren av Memex kunne linke sammen data: På en skjerm kunne brukeren få opp mikrofilmbilder av valgte data, for eksempel en bokside og en avisside. Brukeren kunne da linke sammen disse to bildene, og lagre linken i sin maskin. Når brukeren ved en senere anledning så på boksiden, kunne han trykke på en egen knapp slik at avissiden kom opp på skjermen. På denne måten kunne brukerne lage personlige stier ut av en myriade av informasjon og data. Disse personlige stiene kunne gis til andre mennesker, slik at de kunne bruke dem på sin egen Memex [Bus45, Mau96, Low99].

1.3.2 Ted Nelson og ”hypertekst”

I 1965 presenterte Ted Nelson sin ide om et ”**dokuvers**”. Dette skulle være et univers som inneholdt all informasjon i hele verden. Ingenting skulle bli slettet, og gammel tekst skulle eksistere ved siden av revidert tekst. For å kunne navigere seg rundt i dette ”dokuverset” foreslo han bruk av ”hypertekst”. Prefikset "hyper" brukes for å markere forskjellen mot vanlig lineær tekst. En vanlig lineær tekst leses fra sekvens til sekvens; For eksempel fra det første kapittelet til det siste kapittelet i en bok. I en hypertekst vil brukeren velge neste tekstdel eller dokument ut i fra lenker som ligger inne i foregående tekst. Dette ville gjøre det mulig å følge sin egen vei igjennom dokuverset.

Figur 1-1 Den første versjonen av Xanadu

Nelson selv definerte hypertekst som "en ikke-sekvensiell eller kompleks tekststruktur" og "tekst som ikke lett kan bli skrevet ut".

Nelsons ideer var en videreføring av Vannevar Bush’ ideer. Utviklingen av datamaskiner og informasjonsteknologi har gitt Nelson, i motsetning til Bush, en mulighet til å forsøke å implementere hypertekst i et fungerende system. Han opprettet Prosjekt Xanadu[4] i 1960 for å lage sitt dokuvers, en verden av dype elektroniske dokumenter. Xanadu er fremdeles eksisterende, og kjemper nå for en omkalfatring av Internett, men så langt uten suksess [Mau96].

1.3.3 Tidlige hypertekstsystemer

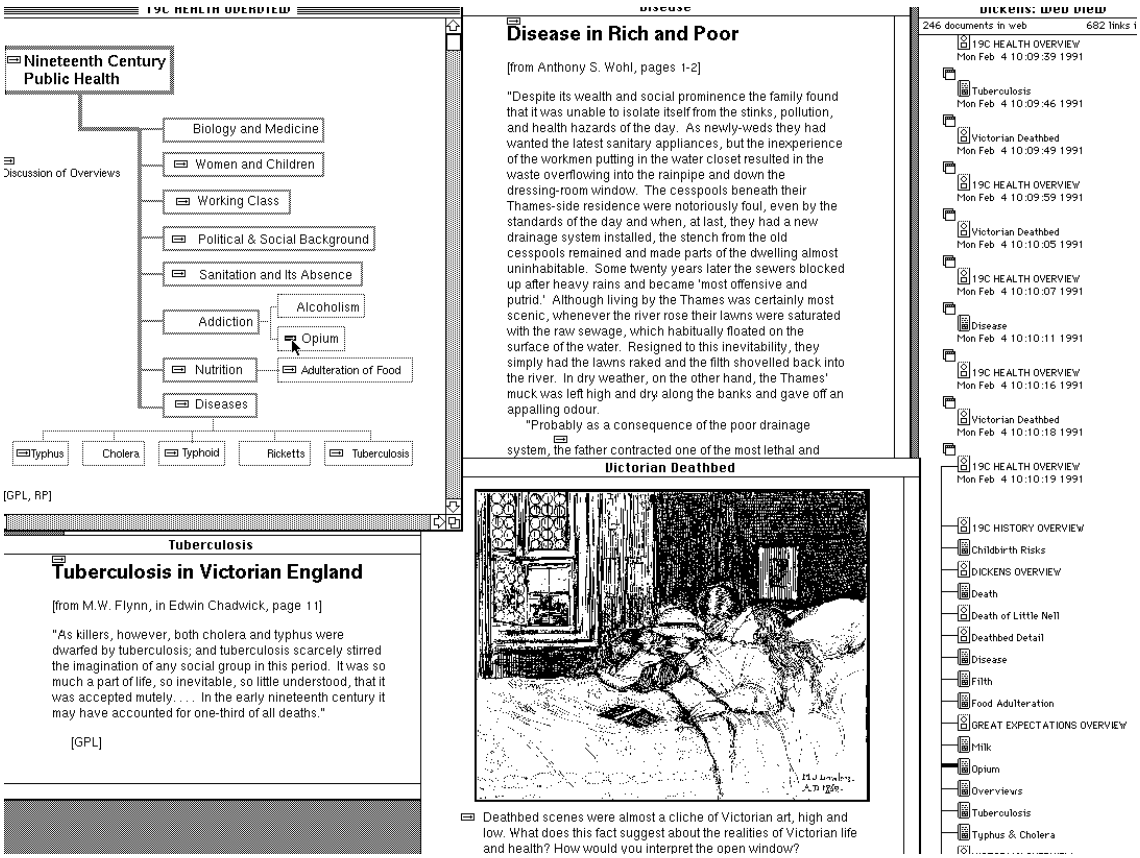
Før WWW eksisterte det flere systemer som for utvikling og bruk av hypertekst. Et par typiske og tidlige eksempler er **Intermedia** og **HyperCard**.

Brown Universitet presenterte sitt hypertekstsystem Intermedia i 1985. Systemet var en samling av verktøy som gjorde det mulig å lage lenker mellom forskjellige typer informasjon (3-D modeller, video, grafikk og tekst). Intermedia tilbød tjenester som teksteditor, grafikk- editor, tredimensjonale kart over objektene, og en timeline editor. Dokumenter som ble laget i Intermedia kunne vise både tekst og grafikk samtidlig i vinduer med rullemenyer.

[4] http://www.xanadu.net/

Hovedideen i Intermedia er at man opererer på samlinger av dokumenter. Oppå disse legger man et nett (engelsk: Web). Dette nettet er et sett av lenker som man bygger opp, og flere brukere kunne legge egne nett på samme dokumenter uten problemer. To typer kart over nettet var tilgjengelig: ”Web View” var et automatisk generert kart, og et kart som forfatter hadde designet spesielt.

Dette gjorde det mulig å lage en modell over forskjellige stier. Intermedia ble blant annet brukt i undervisningen ved universitetet, hvor læreren kunne lage et sett av stier igjennom kursmaterialet. Disse stiene kunne da studentene følge, og i tillegg legge til egne preferanser [Nie90].



Figur 1-2 Et skjermbilde fra Intermedia. Web View er på høyre side.

Intermedia opererte med vinduer som standardenhet, det vil si at hvert dokument kunne få sitt eget vindu. Dette kunne føre til mange åpne vinduer, og gjøre skjermbildet rotete [Sch97]. Som vi ser i Figur 1-2 er det relativt liten plass for selve dokumentinformasjonen.

HyperCard ble introdusert i 1987 av Apple Computers, og ble raskt det mest berømte hypertekstverktøyet i sin tid. En av grunnene til at det ble så populært var at Macintosh lanserte HyperCard gratis sammen med operativsystemet sitt. En annen grunn var at det inkluderte et relativt kraftig og meget enkelt programmeringsspråk kalt HyperTalk. HyperCard var basert på en kort og kortstokk- metafor. Grunnenheten i HyperCard var et kort, og en samling kort ble kalt en kortstokk. Denne metaforen var enkel å lære og å forstå. Brukerne kunne bygge opp en kortstokk, og vise et kort av gangen på skjermen.

En av de mest brukte funksjonene i HyperCard var at man kunne assosiere lenker med knapper. Knapper blir vanligvis aktivert når en bruker klikker på dem med musen, men HyperCard støttet også andre mus-funksjoner: Man kunne aktivere andre hendelser ved at musen gikk over en spesifikk region, eller når en viss tid hadde gått uten brukeraktivitet. Man kunne lett bygge opp et nett av dokumenter på denne måten. Det var også lett å lage fine grafiske skjermbilder [Nie90].

HyperCard ble ikke lengre utviklet eller støttet av Apple fra 2002. En av grunnene til dette var at HyperCard ikke var lønnsomt.

1.4 Internett og World Wide Web

1.4.1 ARPANET

Internett begynte som et militært forskningsprosjekt hos ARPA[5] tidlig på 60-tallet. ARPA var kontrollert av forsvarsdepartementet i USA. De ville bygge ett kommunikasjonsnettverk som kunne omfavne andre nettverk på en desentralisert måte. Ideen var at dette nettverket ikke skulle være avhengig av at alle maskinene fungerte til enhver tid (det var ikke trusselen om et kjernefysisk angrep som var den *primære* motivasjonen). Resultatet var ARPANET. Den første fjernpålogging mellom to datamaskiner skjedde i 1969 mellom Universitetet i California (UCLA) og Stanford Research Institute. ARPANET hadde også støtte for FTP. FTP står for File Transfer Protocol, og er en protokoll som tillater filoverføringer mellom datamaskiner [Mau96]. I 1971 blir også e-post presentert på ARPANET som en tjeneste for alle brukerne. Selv

[5] Advanced Research Project Agency

om disse tjenestene var kommandoorienterte og vanskelige å bruke, ble de snart populære.

1.4.2 TCP/IP

I 1973 ble TCP/IP[6] utviklet av Vinton Verf og Robert Kahn ved DARPA, tidligere ARPA. TCP/IP var en protokoll som gjorde det mulig for forskjellige typer nettverk å kommunisere seg i mellom. Før TCP/IP var kommunikasjon mellom datamaskiner basert på særegne protokoller og teknikker som varierte fra sted til sted. Blant annet benyttet ARPANET protokollen NCP[7]. Fra 1983 måtte alle maskiner som var knyttet til ARPANET benytte TCP/IP. Berkeley UNIX var et meget populært operativsystem ved forsknings – og utdanningsinstitusjoner, og de inkluderte samme år TCP/IP protokollen i sitt system. På samme tid ble protokollen lansert som gratis programvare. Dette gav det tidlige Internett et stort løft. CERN[8] innførte også TCP/IP som standard på sitt eget nettverk mellom 1985–1988.

I 1984 ble ARPANET delt inn i separate deler, MILNET og ARPANET. MILNET skulle fungere som et rent militært nettverk, og ARPANET som et utviklingsnettverk. I 1988 ble de europeiske og amerikanske delene av USENET[9] koblet sammen i et rent IP-nettverk. Denne sammenkoblingen gjorde IP-adressering til de facto standard over hele verden, og CERN ble det største nettstedet i Europa. Flere og flere andre nettverk koblet seg raskt opp til dette større nettverket.

Nettverket ble mer og mer internasjonalt, og "Internett" ble det vanlige navnet. Den vanligste bruksmåten var elektronisk post, nyhetsgrupper og fildeling. Nyttepotensialet var stort, men fremdeles vanskelig. Hvis man ville ha tak i en fil på en annen datamaskin måtte man eksplisitt koble seg opp til den aktuelle maskinen, lete etter den, og benytte kryptiske, tekstlige kommandoer. Internett var fremdeles en greie for spesielt interesserte, studenter, forskere, myndighetene og de militære.[Mau96, And00]

[6] Transmission Control Protocol/Internet Protocol
[7] Network Control Program
[8] Conseil Europĕen pour la Recherche Nucléaire – European Organization for Nuclear Research
[9] Usenet – ”Unix’s users network” er en nyhetsgruppetjeneste. Meget stort og populært.

1.4.3 Winix

Winix var en av de første plattformene som gav et grafisk grensesnitt til Internett. Det startet som et prosjekt i Datasekretariatet i det daværende Kirke og Undervisningsdepartementet. Målet var å lage et verktøy som kunne brukes i skolen for å bedre undervisningen i informasjonsteknologi. Det ble kjørt Unix på tjenersiden, og MS-Windows på klientsiden. Blandingen av MS-Windows og Unix gjorde at brukeren fikk adgang til alle programmer i Winix gjennom et Windows-grensesnitt. Dette gjorde at i tillegg til et felles brukergrensesnitt, fikk man muligheten til å klippe og lime data mellom forskjellige programmer. Det var dessuten mulig å flytte data direkte inn i applikasjoner ved hjelp av Windows DDE (Dynamic Data Exchange). Et eksempel på dette var å ta data som er hentet via terminalemulatoren og sende disse til et regneark eller til et (tekst)dokument. Målet med Winix var ikke å skrive en frittstående programpakke fra bunnen av, men isteden å lage en felles plattform som kunne integrere programmer.

På internettsiden hadde Winix blant annet tjenester som konferansesystem, e-post og terminalemulatorer. E-post var som alle andre programmer tilgjengelig i et Windows-grensesnitt. Terminalemulatoren hadde støtte for skriptspråk som gjorde opp- og nedkobling mot tjeneren enkelt. I tillegg hadde terminalemulatoren støtte for å kjøre Winix-programmer mot tjeneren igjennom telefonlinjer, på samme måte som om PC-en var direkte oppkoblet.

Winix er nå avsluttet som prosjekt i Kirke og Undervisningsdepartementet. Det har ikke vært mulig å få tak i skjermbilder fra Winix.

1.4.4 Tim Berners-Lee og WWW

Tim Berners-Lee ble I 1980 ansatt som programvareutvikler hos CERN. CERN hadde problemer med å gjøre informasjon lett tilgjengelig i dataparken sin. Datamaskinene brukte forskjellige protokoller, og hadde svært forskjellige egenskaper.

Berner-Lees løsning på dette var å lage et system som kunne knytte forskjellige typer informasjon sammen via lenker. For å realisere dette, begynte han i 1989 å utvikle tre teknologier:

- HTML, Hypertext Markup Language. Et format som gjorde det mulig for dokumenter å bli lest og lenket på Internet.
- HTTP, Hypertext Transfer Protocol. Virkeliggjorde en direkte tilgang til dokumentene via deres relative plassering på tjeneren.
- URL, Uniform Resource Locator. Adressen til dokumentet på en nettside.

HTML var en videreutvikling av SGML[10]. SGML var et språk for å lage en logisk dokumentstruktur, og var en internasjonal standard som var utviklet av IBM. SGML er basert på en markering av strukturelle elementer i et dokument, ikke selve dokumentets presentasjon [Rad91].

Berner-Lee kalte det nettverket som brukte disse teknologiene for World Wide Web, og forkortelsen ble WWW, eller W3. Det ble tilgjengelig internt på CERN i 1990, og ble lansert til publikum på Internett sommeren 1991. [Mau96]

Ordene World Wide Web og Internett blir brukt om hverandre. Forskjellen mellom Internett og WWW er at Internett er et fysisk nettverk av nettverker og ble utviklet lenge før WWW. Internett er laget av kabler, datamaskiner og benytter protokoller som TCP/IP. WWW er et abstrakt rom bestående av informasjon. Koblinger på Internett er kabler, koblinger på WWW er hyperlinker. Termen WWW er i dag mindre brukt i dagligtale og i mediene. I praksis brukes nå termen "Internett" som en slags samlebetegnelse som omfatter dem begge.

1.5 Nettleserne

WWW bruker en klient- server modell. Klientene, kalt nettlesere hvis de er ment for interaktiv bruk, er en samling av enkle programmer som kan sende enkle forespørsler til WWW servere. Vi skal se nærmere på dette senere.

[10] Standard Generalized Markup Language

1.5.1 Mosaic

Marc Andressen og Eric Bina lanserte den første grafiske nettleseren Mosaic i 1993. Mosaic kjørte på X-Windows, et grafisk UNIX-system utviklet ved Massachusetts Institute of Technology (MIT). De var to studenter som jobbet på National Center for Supercomputing Applications (NCSA) som tilhørte University of Illinois (UIL). Mosaic kjørte på Macintosh, Windows og Unix. For første gang fikk Internett og WWW et grafisk skjermbilde som man ikke behøvde være ekspert for å benytte. Den ble nedlastet hele 700 000 ganger i 1993, og gjorde at WWW fikk en eksplosjonsartet vekst.

Mosaic hadde et enkelt grensesnitt og var enkel å bruke. Det hvite hus i Washington sendte i november 1993 ut en pressemelding som sa at "Mosaic er den digitale kanonen som har verdensomspennende konsekvenser."[And00]

Den første versjonen av NCSA Mosaic hadde flere udødelige ideer som de fleste andre nettlesere benytter også i dag:

- Knapper for å navigere fram og tilbake, og til en startside(hjem).
- Støtte for forskjellige fonter.
- Støtte for lyd og bilder.
- En liste over nylig besøkte sider.
- Kan lagre og laste dokumenter for fremtidlig bruk.
- En knapp som stopper lastingen av en side (ikonet på NCSA Mosaic 1.0)

1.5.2 Netscape

Andressen tok senere med seg noen av nøkkelfolkene fra NCSA og startet et eget firma i 1994: Mosaic Communications. Dette skapte mye bitterhet hos UIL. Universitetet ble redd for at deres nettleser NCSA Mosaic skulle bli overkjørt av det nyskapte firmaet. UIL lisensierte derfor i 1994 NCSA Mosaic til Spyglass, et firma med nære bånd til universitetet. Samtidlig saksøkte de Mosaic Communications over navnet "Mosaic". Mosaic Communications forandret derfor navn i november 1994 til Netscape

Communications. Netscape fikk raskt verdenshistoriens største børsvekst. I juli 1995 var de verdsatt til 2,2 milliarder dollar, og deres nettleser Netscape Navigator hadde en stund over 90 prosent av markedet. Netscape lanserte også nye og bedre versjoner av nettleseren i rask rekkefølge. [And00]

Netscape kom med en del nye funksjoner, i tillegg til de som allerede ble brukt i NCSA Mosaic:

- Buffering av bilder og dokumenter.
- Datakryptering.
- En identifikator som viste hvor mye nedlasting som gjenstår.
- Autentifisering av servere
- Søkemaskiner innebygd som lenker.
- Støtte for Java og JavaScript.

1.5.3 Internet Explorer

Det Seattle-baserte firmaet Microsoft, styrt av Bill Gates, ble oppmerksom på at Mosaics hadde et stort potensial. Microsoft var redd for å miste mulighetene i den gryende interessen for WWW. Ledet av Brad Silverberg prøvde Microsoft å kjøpe nettleseren BookLink fra det lille firmaet Internetworks. BookLink hadde et par nye finesser: Det var mulig å benytte nettleserens direkte fra andre programmer. BookLink brukte også mye av Windows OLE, teknikken som gjorde det mulig å redigere og lese Windows-dokumenter uavhengig fra programmene som skapte dem (for eksempel kan man redigere et Paint-bilde i Word, uten å måtte kalle Paint). Dessverre for Microsoft ble BookLink solgt til America Online (AOL). Microsoft kjøpte da lisens på Spyglass' nettleser i 1994. og begynte ut i fra det å utvikle sin egen nettleser, Internet Explorer (IE).

IE introduserte blant annet ”style sheets”, noe som gjorde det mulig å se dokumenter med en egenprodusert preferanse [And00, Mau96, Low99]. Versjon 3.0 av IE som ble lansert i 1996 hadde støtte for skriptspråket Visual Basic.

1.5.4 Nettleserkrigen

Både Microsoft og Netscape bygde sine nettlesere med et lignende grafisk grensesnitt som NCSA Mosaic, men la etter hvert til mange nye funksjoner:

- Knapper for å fornye lasting, skrive ut, åpne og finne dokumenter
- Nyere og sikrere sikkerhetsprotokoller
- Bedre grafikk
- Raskere intern tegning av dokumenter.
- Støtte for video.
- Større for flere forskjellige filtyper.
- Integrering av e-post og nyhetsgrupper.
- Videokonferanser.

Selv om NCSA Mosaic hadde relativt mange brukere fra begynnelsen av, hadde den et grunnleggende problem: Den hadde ikke store nok ressurser bak seg for å støtte opp videreutviklingen raskt nok. Microsoft og Netscape hadde store utviklingsteam og ressurser, og lanserte nye og bedre versjoner av sine nettlesere i kapp med hverandre. Mange flere aktører meldte seg også på i kampen om nettlesermarkedet, både små og store. En av de største er AOL. Norge har også sitt bidrag med Opera, utviklet av Opera Software[11].

Lanseringen av IE 4.0 i 1997 var et vendepunkt for Microsoft. Fra da av begynte IE gradvis å ta over nettlesermarket fra Netscape.

Microsoft integrerte IE inn i operativsystemet Windows98SE i 1999. Begrunnelsen bak dette var Bill Gates' ideologi om "Internett ved din fingerspiss": Microsoft skulle bryte ned barrieren mellom brukerens egen datamaskin og Internett. For brukeren skulle det være det samme om filene lå på hans egen harddisk eller på en server i et annet land. Dagliglivet skulle bli fullt av små datamaskiner som organiserer kjøleskapet, lyset, bilen og så videre. Disse skal snakke sammen, og en person kan dermed få utvidede muligheter til å få tak i og behandle informasjon.

[11] www.opera.com

Senere versjoner av Microsofts operativsystemer har videreført denne ideen ved å integrere e-postklienter, nyhetsgruppeklienter og annet inn i IE og Windows. Denne integreringen førte til at Microsoft ble saksøkt i USA for brudd på monopollovgivningen av blant annet AOL. Bakgrunnen til dette er at AOL kjøpte opp Netscape i 1999. Tvisten med AOL ble avgjort med en avtale i 2003, med det resultat at Microsoft skal gi AOL 700 millioner dollar og gi AOL lisens på IE i 7 år.

I juli 2003 ble det også klart at AOL ikke ville videreutvikle Netscape som et kommersielt produkt. Utviklingen av nettleseren vil fortsette, men nå som en del av det prosjektet som utvikler nettleseren Mozilla. Per oktober 2003 har IE en markedsandel på over 90 % [12].

1.5.5 Videre utvikling

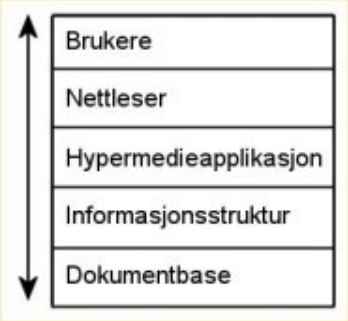
Bill Gates’ ide om "Internett ved din fingerspiss" har ikke krystallisert seg ordentlig, og resultatet har latt vente på seg. Vi har pr. i dag mange "duppedingser" som mp3-spillere, mobiltelefoner som er tilknyttet til Internett og digitale kameraer som kan sende bilder til andre maskiner. Men det er en lang vei før noe av dette er bygd inn i hverandre, slik at brukeren faktisk har Internett på sin fingerspiss. Kanskje man kan sammenligne dagens situasjon med den som eksisterte før TCP/IP ble standard: Med mange forskjellige teknologier og standarder som bekjemper hverandre. Uansett er dette en ide som vil være verd å følge med i fremtiden.

[12] http://www.upsdell.com/BrowserNews/stat.htm

2 Teori og strukturelle utfordringer

2.1 Innledning

Dette kapittelet vil presentere oppbyggingen av hypermedia, og gi en generell bakgrunn til navigeringsproblematikken bak den. Hypermedia består av mange lag av komponenter, så for enkelhets skyld har vi delt dem opp i hierarkiske deler som vist i Figur 2-1.



Figur 2-1 Hypermedias hierarki

Som bruker på toppen av dette hierarkiet ser man alle disse lagene på en gang. Man kan forestille seg hierarkiet som et lag med gjennomsiktige vinduer. En bruker ser på dokumentbasen igjennom informasjonsstrukturen, hypermedieapplikasjonen og nettleseren.

Vi vil gå igjennom disse delene hver for seg, og vise hvordan de sammen utgjør det univers brukerne navigerer seg i. Lærdommen fra dette kapittelet vil vi ta med oss til kapittel 3, hvor vi i mer detalj vil undersøke forskjellige aspekter ved grafiske kart.

Fokus ligger på navigasjon i hypermedia, ikke hypermedia i seg selv.

2.2 Hva er hypermedia?

Som nevnt i 1.3.2, er det forskjell på vanlig tekst og hypertekst. Hypermedia blir ofte beskrevet som den moderne tidens hypertekst, med den eneste forskjellen at bilde, video og lyd også kan brukes på samme måte som tekst. Det finnes mange definisjoner:

"*Hypertekst er tekst som ikke er lineært begrenset*"

"*Hypermedia and Hypertext tend to be used loosely in place of each other. Media other than text typically include graphics, sound and video*"

World Wide Web Consortium, 'Hypertext Systems', April 1995.

"*Hypertext: …is a web of possibilities, a web of reading experiences. …Hypertext is the language of exploration and discovery*"

Charles Deemer, “What is hypertext,” 1994.

I stedet for å prøve å finne den perfekte definisjonen på hypermedia, kan det være fruktbart å se på hva som karakteriserer hypermedia:

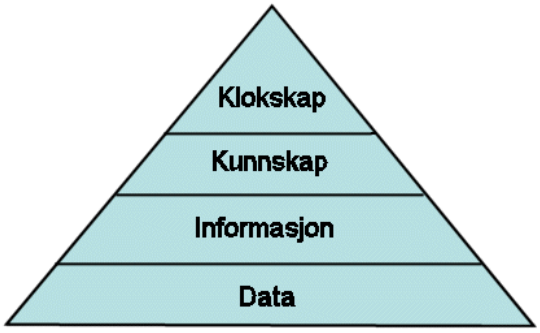
- Hypermedia skal kunne benyttes til både å skrive og lese informasjon
- Informasjonen er av en ikke-sekvensiell struktur, og informasjonen kan derfor følges via flere alternative ruter.
- Informasjonen kan følge naturlige assosiasjoner fra en informasjonsenhet til en annen.
- Informasjonen kan være strukturert i hierarki.
- Informasjonen, eller deler av informasjonen skal kunne deles mellom flere brukere.
- Informasjonen ligger helst i en database. Med database mener man en samling av data, og disse dataene kan nåes på en eller annen måte. Det forutsettes ikke at dataene er strukturert på en spesiell måte, eller at de følger en spesiell orden [Low99, Sta1998].

2.3 Forholdet mellom data, informasjon, kunnskap og klokskap

Hypermedia handler om informasjon. Konseptet "informasjon" er ganske vagt, men vi vil her forklare det i konteksten hypermedia:

Lowe hevder at et hypersystem kan presenteres som en pyramide hvor data er nederst, fulgt av informasjon i midten og kunnskap på toppen. Vi vil tillegge denne pyramiden et trinn til, nemlig klokskap. Klokskapen ligger på toppen, over kunnskap. Dette er fordi klokskap er måten kunnskapen blir brukt på.

- Klokskap: måten kunnskapen blir brukt på.
- Kunnskap: En personlig samling informasjon som er integrert på en slik måte et den kan bli brukt for videre forståelse og analyse av ny informasjon.
- Informasjon: Oversettelse av data innenfor en kontekst. Denne oversettelsen er bestemt av tidligere kunnskap, og av omgivelsene.
- Data: En samling symboler og tekst ("ting som eksisterer som et fremkomstmiddel for informasjon.")



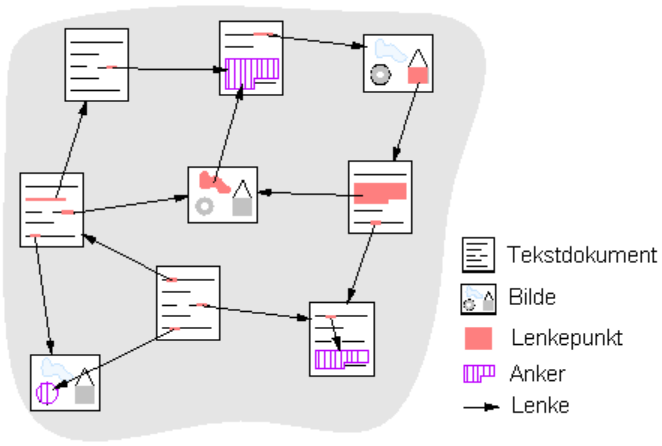
Figur 2-2 Kunnskapshierarki

Ut i fra sin pyramide trekker Lowe følgende konklusjoner:

- I et hypermediesystem har ikke data noen verdi i seg selv. Derfor vil faktorer som hindrer en effektiv presentasjon av dataene ha en direkte negativ innvirkning på brukerens mulighet til å tolke dataene.
- Data blir forstått i den kontekst de presenteres
- Informasjon blir forstått i fra det miljø de presenteres i.
- For å kunne trekke ut noen brukbar kunnskap fra dataene, bør dataene følge en meningsfull struktur [Low99].

2.4 Hypermediebasen

Hypermediebasen består hovedsakelig av hyperdokumenter, noder og lenker. Hypermediebasen er altså selve informasjonsbasen som et hypermediesystem arbeider oppå [War00].



Figur 2-3 Eksempel på et hyperdokument

Noder er individuelle biter av informasjon. Noder kan være bilder, lyd, en bit tekst, video og så videre [Low99]. Vanlig praksis er at disse bitene bør være så små at de ikke kan deles opp mer uten å miste sin opprinnelige mening. De bør kunne stå for seg selv, det vil si at ytterlig informasjon ikke skal være nødvendig for å forstå innholdet i noden. I tillegg bør hele noden få plass på skjermen på en gang.

Lenker knytter noder til hverandre. På denne måten kan forfatterne bygge opp et nettverk av informasjon. Man kan skille mellom to typer lenker: Referanselenker og strukturelle lenker. Referanselenker er de kryssreferansene som er typiske for hypermedia, og lenker mellom selve innholdet i hypermediebasen. Typiske eksempler er lenker som gir mer informasjon om et emne. Strukturelle lenker viser selve strukturen til hypermediebasen. Typiske strukturelle lenker er lenker til startsiden, og lenker til sider som ligger i nivået under den gjeldende siden. [Mau96, Nie99]

Lenker kan være hardkodet inn i nodene, eller være abstrahert inn i en serverbasert løsning. Vi vil komme tilbake til dette senere.

Ankret er det punktet eller teksten som representerer lenken. Enhver lenke har to ankre, et som går til og et som går fra. Ankere blir vanligvis representert i en annen farge enn det omliggende innholdet. Lenker kan være i to tilstander, besøkt eller ikke-besøkt. Forskjellen vises vanligvis ved at fargen på ankeret forandres etter et besøk. De fleste nettleserne (også IE) har en standardvisning av ankere hvor ikke-besøkte ankere er blå, og besøkte er lilla. Ankere har uansett tilstand en strek (underline) under. Siden IE har over 90 prosent av markedet, er dette de facto standard for lenkevisning.

Eksempel på anker: ”Vi har mer om informasjon om neshorn”.

Stier er et sett av lenker som binder noder sammen på en sekvensiell måte. Stier er ofte brukt som et navigasjonshjelpemiddel. Brukeren følger stien igjennom hypermediesystemet for å oppnå et spesifikt mål. Målet kan for eksempel være en oversiktspresentasjon av et hypermediesystem, eller å lære et sett av tjenester på en lineær måte.

Hyperdokumentet er en samling noder og lenker. Hyperdokumentet omfatter gjerne et selvstendig, begrenset tema. I praksis vil dette si at hyperdokumentene er egne filer. Det mest brukte filformatet er HTML-formatet.

Nettet (engelsk: Web) er en abstraksjon av lenkene som finnes mellom hyperdokumentene. Man kan si at nettet er den samlingen av lenker som man kan benytte for å navigere i hypermediebasen. Denne abstraksjonen gjør det mulig for å ha flere nett på samme hypermediebase. Denne funksjonen er særlig nyttig i et flerbrukersystem.

Metadata[13] kan tilhøre både ankre, lenker, noder og hyperdokumentet, avhengig av hypermediesystemet. Formålet til metadata er å klassifisere og beskrive det tilhørende objektet. Tjenester som er vanlige å utføre på metadata er:

- Støtte søk på informasjon
- Indeksering av hypermediebasen

[13] Metadata betyr ”data om data”. Det er attributter som beskriver et objekt.

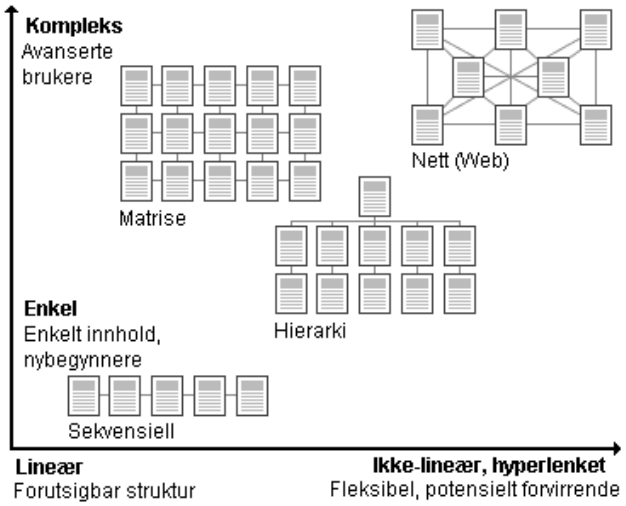
- Gi ekstra informasjon ved lenking mellom noder.

2.5 Informasjonsstruktur

Før man utvikler et hypermediesystem har man gjerne en ide om hvordan skal presentere hypermediebasen igjennom en informasjonsstruktur. Som vist i 2.4 består hypermediebasen av nettverk av noder og lenker. Disse nettverkene gjør det mulig å utnytte assosiasjoner mellom konsepter. Vanligvis vil man bare gjøre noen få assosiasjoner eksplisitte, siden man kun vil prøve å formidle visse ideer. Informasjon som er relatert i forhold til et visst konsept, vil forholde seg til annen informasjon avhengig av konteksten. For eksempel er Michael Gorbatchov og Ronald Reagan relatert i forhold til konteksten ”internasjonal politikk i 80-årene”, men ikke relatert i forhold til konseptet ”berømte sovjetiske statsmenn”.

Struktureringen av informasjonen bestemmer i stor grad hvilke tjenester som man kan benytte, og om tjenesten er funksjonell eller ikke. Ofte vil strukturen bestemmes av hvilke typer av lenker man vil bruke; en hierarkisk struktur er god til å vise strukturelle lenker, mens en nettverksstruktur er god til å vise referanselenker.
Informasjonsstrukturer kan bli klassifisert på forskjellig måte, men det er vanlig er å dele dem i 4 deler: Sekvensielle, hierarkiske, nettverk og matriser. Det er viktig å merke seg at det ikke er noe totalt skille mellom de forskjellige typene. Et hypermediesystem kan benytte flere typer samtidlig. Forskjellige strukturer kan også benytte samme hypermediebase.

Sekvensiell struktur er meget god å bruke hvis man vil bevare en original lineær struktur fra et dokument (for eksempel en bok). Hvis informasjonen opprinnelig var laget sekvensielt, er det god praksis å fortsette denne strukturen. Hvis ikke, kan en del informasjon falle ut ifra den originale konteksten. For eksempel kan forfatteren ha gitt leseren en helt ny grunnforståelse av et konsept i kapittel 2, og presentere konseptet på nytt i kapittel 3. Hvis leseren da ikke leser kapittel 2 før han leser kapittel 3, kan det gi grobunn for misforståelser.



Figur 2-4 Forholdet mellom ulike informasjonsstrukturer

En sekvensiell struktur er forutsigbar og enkel å forstå, da alle som kan lese og skrive vil være kjent med den. Treningsmateriale kan tjene på en sekvensiell struktur, da man kan forsikre seg at kurset blir vanskeligere etter hver. I tillegg kan forfattere stille spørsmål på forskjellige punkter, for eksempel ved hvert kapittels slutt.

Hierarkisk struktur benytter seg av ideen om at alle noder og lenker befinner seg på forskjellige spesifikke nivåer i forhold til hverandre. Øverst i hierarkiet er toppnoden, og alle andre noder kan nåes ut i fra denne. Bøker er ofte delt opp i kapitler og underkapitler, og denne strukturen kan kopieres i et hypermediesystem. En leser kan gå til en indeks, og velge et punkt å lese videre fra. Strukturen er veldig lik Windows Utforsker, hvor man kan gå opp eller ned i et hierarki ved å klikke på kataloger. Hierarki kan være nyttig i utdannelsesøyemed, da informasjonen kan bli mer detaljert jo lengre ned i strukturen man går.

Det kan være et problem å utvikle en hierarkisk struktur hvis informasjon ikke har et logisk nivå/plass å bli plassert på. I tillegg kan den være tungvint å bruke hvis strukturen er meget dyp. En bruker kan da risikere å måtte klikke seg igjennom mange lag for å finne helt spesifikk informasjon.

Nettverksstruktur er det som mest typisk fanger inn grunnideen bak hypermedia. Strukturen kan ha strukturelle lenker, men baserer seg hovedsakelig på referanselenker. Den er ikke-sekvensiell, men binder sammen relaterte noder innenfor det generelle informasjonsrommet. Det er få restriksjoner ved bruk av denne strukturen. Ideen er å

imitere assosiativ tenking og en fri strøm av ideer. Lenker kan like lett gå til informasjon som ligger utenfor nettstedet. Et leksikon på Internett kan for eksempel tjene på å bruke en nettverksstruktur, da noder kan bli lest i forskjellige kontekster, og brukeren lett kan hoppe til nærmere forklaringer av ord og uttrykk.

Denne strukturen er ironisk nok den mest konseptuelt utfordrende for et nettsted da det er vanskelig for brukeren å forstå og forutse organiseringen.

Matrisestruktur kan være brukbar hvor man har en flerdimensjonal kategorisering av informasjonen. For eksempel blir den mye brukt i industrien: den horisontale aksen kan være forskjellige mekaniske problemer, og den vertikal aksen kan være informasjon om symptomer, løsninger og prosedyrer. Dette vil resultere i en enkelt celle som for eksempel vil gi informasjon om symptomer på problemer med girkassen. Matrisestrukturen blir ofte brukt på toppnivå i en hyperstruktur, og kan gi en god indekseringsmekanisme for lavere nivåer.
[Low99]

For alle informasjonsstrukturer er det viktig at implementeringen er konsis og korrekt i alle henseender. Store hypermedieapplikasjoner kan ha flere forfattere, mange tusen noder og et vidt spekter av informasjon som skal passe inn. Dette kan medføre en høy kompleksitet hvis man ikke er nøye med å plassere de fysiske dokumentene på logiske plasser.

2.6 Navigeringstjenester

For å benytte hypermediebasen trenger vi tjenester som vi kan utføre på den. Disse tjenestene kan eksistere i hypermedieapplikasjonen eller i nettleseren. Ikke alle tjenester som er listet under finnes i alle hypermedieapplikasjoner. Men Internett har gjort de fleste av disse til en standard ved at de er tilbudt i de fleste nettlesere i dag. (Vi skal komme tilbake til forholdet mellom hypermedieapplikasjoner og nettlesere senere).

2.6.1 Søkemotorer

Søkemotorer er en ting som kan hjelpe brukeren til å finne fram til ønsket informasjon [Con87]. Tjenester som søkemotoren kan tilby kan være søking etter ord, innhold, likhet, eller en kombinasjon av disse. Den vanligste tjenesten er søking etter ord, gjerne med mulighet for boolske variabler[14] eller jokertegn[15]. Søkemotorer indekserer hyperdokumentene i hypermediebasen etter en matematisk formel, og bygger opp en indeks med unike identifikatorer for hvert dokument. Denne indeksen kan da brukeren benytte ved å taste inn en eller flere termer. Det er også vanlig å indeksere metadata.

Hypermediebasen som søkemotorer bruker kan være avgrenset til en lokal base, et intranett, eller deler av Internett. En søkemotor er særlig nyttig hvis mengden informasjon er så stor at brukeren ikke har anledning til å bli kjent med hele hypermediebasen [Nie90]. Bruk av søkemotorer på Internett er svært vanlig i dag. Søkemotoren Google[16] indekserer per oktober 2003 1,3 milliarder hyperdokumenter på Internett, og svarer på mer enn 100 millioner søk per dag.

2.6.2 Innholdsliste

De fleste hypermedieapplikasjoner har en innholdsliste. Av og til blir bare hovednivåene vist, gjerne på nettsteder som er så store at en fullstendig liste ville bli for plasskrevende. Ved valg av et nivå eller tema, vil listen gjerne vise et mer detaljert bilde av innholdet. Som regel blir de vist på venstre side i nettleseren. Disse listene fungerer på samme måte som innholdsfortegnelsene i bøker, og er derfor lett å forstå. En indikering av hvor man befinner seg i indeksen kan vises ved at det gjeldende punktet blir fremhevet eller markert på en eller annen måte.

Noen nettsteder benytter såkalte rullegardinmenyer, som gir tilnærmet samme funksjonalitet, og som er mindre. Men det er vanskelig å forandre farge på besøkte sider

[14] En variabel som kan ha to verdier: sann og usann. Blir gjerne indikert i søkemotorer med tegnene for pluss og minus. Eks: Et søk på navnet Ole - Hansen vil kunne resultere i for eksempel Ole Olsen, Ole Karlsen, men ikke Ole Hansen.

[15] Et vanlig jokertegn er *, som vanligvis betyr at resten av ordet kan være hva som helst. Eks.: Et søk på ordet inter* i en søkemotor vil kunne resultere i f.eks. Internett, interessert, intervall, interresse, osv. som søkeresultat

[16] www.google.com. Navnet Google kommer fra det engelske tallet gogool, som er et ett-tall etterfulgt av hundre nuller. Dette skal gjenspeile selskapets intensjoner om å indeksere hele Internett.

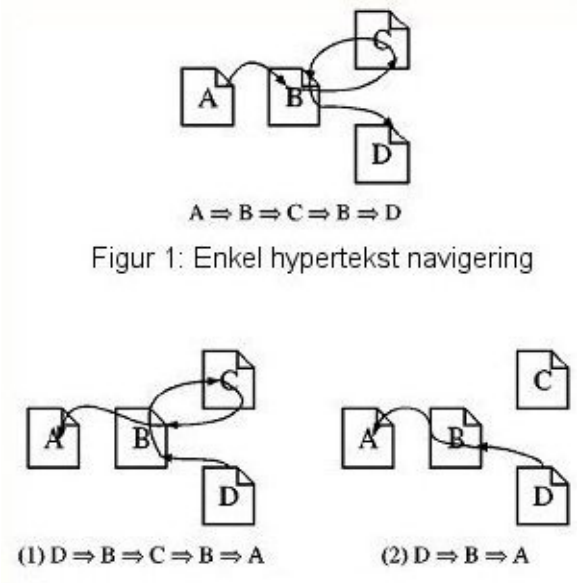
i en slik meny. I tillegg vil en rullegardinmeny ikke eksplisitt vise alle valg. Brukeren må hente den fram med musen, og det er lite ønskelig.[Nie99]

Selve menyen tar opp en del av plassen på skjermen, og forfatterne må avveie plassbehovet mellom selve innholdet og menyen.
Forfatter må avveie plassbehovet mellom innholdslisten og selve dokumentet. Dette gjelder særlig hvis listen vises konstant i en egen ramme uansett hvor brukeren befinner seg i nettverket.

2.6.3 Historien

Hypermedieapplikasjonene registrerer (og lagrer til en viss grad) hvilke nettsider en bruker har besøkt. Informasjonen blir lagret i en liste som vi kaller historien. Den bakenforliggende logikken er at applikasjonen bygger opp en liste bestående av URL fra alle noder og lenker etter hvert som brukeren leser dokumentene. Hver URL får også en dato og et klokkeslett for det aktuelle tidspunktet. Historien gjør det mulig for brukerne å returnere til tidligere besøkte noder. Denne tjenesten er viktig for brukerne, da den gir dem en tillit til å utforske hypermediesystemet uten å være redd for å ikke kunne returnere [Mau96].

De fleste hypermedieapplikasjoner har knapper for å gå fram og tilbake i historien. Et klikk på **tilbakeknappen** tar brukeren et steg tilbake, og omvendt med **framknappen**. Naturligvis vil ikke framknappen fungere før brukeren har besøkt en side og gått et steg tilbake igjen. I noen hypermedieapplikasjoner vil historien bli slettet når sesjonen avsluttes. Tilbakeknappen er en meget brukt tjeneste. Linda Tauscher og Saul Greenberg gjorde en statistisk undersøkelse over bruk av nettlesere i 1997, og fant ut at tilbakeknappen stod for over 30 prosent av alle knappetrykk hos nettleserne. Til sammenligning utgjorde framknappen bare 1 prosent [Tau97].



Figur 2-5 Forskjellige metoder for bruk av historien

De fleste hypermedieapplikasjoner implementerer historien som en endimensjonal liste. Dette skaper problemer da den reelle historien som bygges opp av en bruker når han traverserer WWW ikke er endimensjonal. Som vi ser i **Feil! Fant ikke referansekilden.** finnes det forskjellige metoder for å gå tilbake i historien. Løsning 1 benytter en kronologisk analogi, man reverserer historien til brukeren. På denne måten er man sikret at alle sider som har vært besøkt vil bli vist. Denne løsingen er konsis, og har den fordelen/ulempen at man også vil vise noder som ”stikker ut” (C) i fra stien. Løsning 2 benytter en oppgaveorientert analogi. Historien forandres etter hvert som brukeren klikker seg videre, og noder kan forsvinne (node C). Dette er et problem, da historien mister informasjon, uten at brukeren blir gjort oppmerksom på det. Både Netscape og IE benytter den oppgaveorienterte analogien.

Historieliste er en tjeneste som lister opp de siste besøkte hyperdokumentene. Det er vanlig å vise tittelen på hyperdokumentet, ikke URL. I Internett Explorer fungerer historien ved at man trykker på en egen historieknapp, og en liste over historien presenteres i en egen ramme på venstre side av nettleseren. Denne tjenesten gjør det mulig å hoppe direkte til sider som ligger langt tilbake i historielisten, uten å måtte trykke på tilbakeknappen flere ganger.

2.6.4 Bokmerker/Landemerker

Landemerker er enveis lenker til et spesifikt punkt i nettverket. Det er vanlig å ha landemerkene permanent synlig for brukeren. Eksempel på vanlige landemerker er lenker tilbake til startsiden [Bie00, Nie99]. Det er vanlig standard at ethvert hyperdokument i et nettverk har en lenke tilbake til startsiden. Siden eksterne sider kan lenke til nivåer dypt nede i en hypermediebase, er det god praksis at hvert dokument gir en potensiell lenke til toppnivået.

Landemerker er skrevet av forfatter av nettverket, mens **bokmerker** blir skapt av brukeren. Bokmerker er i bunn og grunn personlige landemerker. De lagres i hypermedieapplikasjonen, vanligvis i URL format. Brukeren kan lage flere bokmerker. I nettleserne lages bokmerker ved at brukeren markerer bokmerket når det aktuelle dokumentet er i fokus. Når han senere vil gå til et bokmerke, velges det aktuelle bokmerket ut i fra en liste over alle tilgjengelige bokmerker. Som i historielisten er det ikke URL som blir vist, men tittelen på dokumentet. Dette er mer brukervennlig, da en URL kan være kryptisk og ikke gi noe særlig informasjon. (For eksempel når et hyperdokument blir automatisk navngitt av serveren. Følgende URL er til en artikkel på aftenposten sin nettavis:

http://www.aftenposten.no/nyheter/nett/article.jhtml?articleID=656046

I motsetning er tittelen mer forklarende: ”Norge høyt på internasjonal nettstatistikk”.

2.6.5 Brødsmuler

Brødsmuler følger samme analogi som i eventyret Hans og Grete. For hver side eller nivå som man besøker, vil man etterlate et lite spor som man senere kan gå tilbake til. Denne funksjonen er meget lik historiefunksjonen i nettleserne. Brødsmuler fungerer kun i en informasjonsstruktur som benytter hierarki. Dette er fordi hver side må tilhøre et nivå i strukturen. Brødsmuler viser nivået siden ligger på, ikke selve siden.

Figur 2-6 Eksempel på Brødsmuler i Opsys

Som man kan se i Figur 2-6, har brukeren besøkt først seksjonen "Innhold", så "Kapittel 1", og så videre. Dette gir brukeren både et landemerke å forholde seg til, og en pekepinn på hvor han befinner seg i informasjonsstrukturen.[Nie99].

2.6.6 Stier

Vi har allerede sett at stier knytter sammen lenker igjennom nettverket. De gir en kontekst for å se og forstå en serie av noder. Forfattere kan lage stier som fokuserer på forskjellige aspekter, eller er laget med tanke på forskjellige brukere (nybegynnere, eksperter osv.). Stier kan også selv inneholde stier, som brukeren kan velge å følge. Det er vanlig å bruke brødsmuler i stiene for å vise brukerens posisjon, hvor langt inn i stien han er, og hvor mye det er igjen [Bie00].

2.6.7 Nettkart

Nettkart gir et grafisk oversiktsbilde over nettverket, vanligvis med nodene som ikoner og lenkene som piler/streker mellom nodene. Det er vanlig å vise tittelen til det tilhørende dokumentet i noden. Nettkartene kan vise ulike detaljnivåer og omfang av et nettverk. Hensikten med nettkartene er å vise nodenes kontekst, og å redusere desorientering. Brukeren hopper direkte til et dokument ved å trykke på noden [Bie00].

Vi skal komme tilbake til nettkart i Kapittel 3.

2.7 Nettapplikasjoner

Den første generasjonen hypermedieapplikasjoner kom som regel på CD-ROM, og var gjerne ikke særlig viktig eller essensiell for noen organisasjon. Informasjonen var heller ikke ment å bli oppdatert [Ros90]. Internett har forandret dette. I dag når vi snakker om hypermedia snakker vi om Internett, og når vi sier hypermedieapplikasjoner mener vi egentlig applikasjoner som fungerer på Internett. Dette er ikke å si at hypermedieapplikasjonene er blitt utdatert og forbigått, men heller utvidet til å gjelde Internett. I dag finnes det nettsteder som har en del av den samme funksjonaliteten som eksisterte i de tidligere hypermedieapplikasjonene. Skriptspråk som JavaScript og Perl,

og introduksjonen av plattformsuavhengige programmeringsspråk som Flash og Java har gjort det mulig å støtte avanserte funksjoner på nettsteder.

[Low99 s.74] mener de tradisjonelle hypermedieapplikasjonene har en viss felles funksjonalitet. Vi lister opp de viktigste av disse:

- Støtte for assosiasjonsbasert navigasjon, det vil si bruk av referanselenker.
- Et kart som gir brukeren en forståelse av strukturen i nettverket, og hans relative posisjon.
- En måte å fortelle brukeren hvor han har vært. Dette blir ofte gjort igjennom historielister, bokmerker og landemerker.
- Søkemotorer
- En mulighet til å forandre den underliggende hypermediebasen (sletting, redigering og opprettelse av dokumenter).

Som vi ser kan et nettsted inneholde mye av denne funksjonaliteten (Det er ikke vanlig at brukeren kan forandre hypermediebasen igjennom nettleseren, selv om det er mulig). Noen funksjoner kan bli ivaretatt av nettleseren. Andre kan bli ivaretatt av programmer i Java, god dokumentdesign og skriptspråk. For å dekke både begrepet hypermedieapplikasjon og nettsteder som fungerer som hypermedieapplikasjoner, lanserer vi begrepet **nettapplikasjoner**. Vi hevder ikke at enhver hjemmeside er en nettapplikasjon. Visse forutsetninger må være oppfylt:

- Nettstedet blir vedlikeholdt og oppdatert over flere år.
- Brukeren gies en metode for å lage et personlig oppsett.
- Det må inneholde en viss standard og mengde informasjon. Denne informasjonen må langt overgå nivået for reklame og brosjyrer.
- Det har tjenester for søking etter ord, og egendefinerte bokmerker og landemerker.
- Nettstedet har en strategi og klare formål.

Vi vil ikke spesifisere disse forutsetningene mer detaljert, da hvert tilfelle bør vurderes separat. Som eksempel på nettapplikasjoner vil vi nevne Opsys og Innsida[17].

2.8 Lost in hyperspace

2.8.1 Behov for videre utvikling

Georgias Tekniske Institutt, hadde i 1998 sin tiende undersøkelse om trender på Internett. Gruppen som utførte denne forskningen heter GUV[18]. I følge GUV var det bare 3.7 % av 123 personer som svarte at det var et problem at de gikk seg vill i WWW[19].

Bieber mener dette tyder på at brukerne har tilpasset seg de forskjellige navigeringstjenestene som historieliste, landemerker og så videre. Men bruken av teknologien er allerede bestemt av WWWs innebygde funksjonalitet, og brukerne vil derfor prøve å gjøre det meste at av de funksjonene de har tilgjengelig. En videre utvikling av hypermedia behøver ikke strengt tatt være nødvendig, men kan likevel øke forståelsen av brukerens posisjon i et hypermediesystem [Bie97].

Vi hevder i tillegg at selv om man ikke går seg vill på WWW, er ikke dette en brukbar måte å bestemme om WWW er funksjonelt eller ikke. En turgåer vil ikke kun bedømme et kart ut i fra at han ikke går seg vill med det, men også om kartet gir ham en god sjanse til å nå en destinasjon på en effektiv måte.

GUV viste i sin undersøkelse at:

- 45.4 % hadde problemer med å finne ny informasjon. Med ny informasjon menes det at brukerne vil vite noe om et emne de er ukjent med.
- 30 % hadde problemer med å finne en side som de viste eksisterte.
- 16.6 % hadde problemer med å finne en side de allerede hadde besøkt.

[17] NTNUs intranet portal. http://innsida.ntnu.no
[18] Graphics, Visualization & Usability Center, Georgia Institute of Technology
[19] http://www.gvu.gatech.edu/user_surveys/survey-1998-10/

- 7.4 % kunne ikke visualisere hvor de hadde vært, og hvor de kunne gå. Dette gjelder for eksempel å se en del av et nettsted[20].

Det er tydelig at hypermedia og Internett ennå ikke har nådd sitt fulle potensial. Mange problemer kan løses ved bedre bruk av eksisterende metoder. Blant annet er det mange nettsteder som er dårlig designet, og bruker feil informasjonsstrukturer på hypermediebasen. Vi skal ikke komme inn på temaet dårlige nettsteder her.

2.8.2 Desorientering og kognitiv overbelastning

Det er vanligvis vanskeligere å navigere i store hypermediebaser enn små. Brukeren vil ikke nødvendigvis klare å få en fullstendig oversikt over innholdet. Dette er fordi skjermen ikke har plass til en visning av alle dokumenttitlene og lenkene på en gang. I tillegg er det vanskelig å forstå strukturen til nettverket, og hvor man selv befinner seg i den. Dette blir enda vanskeligere på det generelle Internett. Det finnes ikke noen fastlagt struktur som alle følger, selv om visse trekk er mer vanlige enn andre (innholdslister, søkemotorer og landemerker). Selv om man finner fram til siden man vil nå, kan videre navigering ut i fra den også være problematisk. Noe av problemet er at antallet linker mellom dokumenter har en tendens til å vokse eksponentielt med antall dokumenter. Et nettverk med 1000 dokumenter vil kanskje ha 100 ganger flere linker enn et nettverk med 100 dokumenter. Og det er ikke bare antallet linker som vokser, den bakenforliggende strukturen vil også bli mer kompleks, og skape problemer både når det gjelder navigasjon, og ved vedlikehold av systemet.

Jeff Conklin hevder at dette gir er to fundamentale problemer ved navigering i hypertekst: **desorientering** og **kognitiv overbelastning**. Med desorientering mener man tendensen til å miste stedsoppfatning og retning i et ikke-lineært miljø. Brukeren vet ikke hvor han er, og vet ikke hvordan han skal komme seg dit han vil. Brukeren har rett og slett rotet seg bort [Con87].

[20] Tallene er tatt fra en av flere grupper som GUV spurte, men de fleste gruppene har prosentandeler som samsvarer med listen.

Kognitiv overbelastning refererer til brukerens mulighet for å følge lenker som er indirekte relatert til den gjeldende oppgaven, enten med vilje eller ved distraksjon. Dette gjelder også når brukeren må følge flere stier for å besøke så mange noder i et nettverk som mulig. Vanskeligheten er å velge mellom flere lenker, særlig når brukeren ikke er kjent med nettverket, eller er en nybegynner [Bie97].

Conklin hevdet at for å løse problemet med desorientering var det nødvendig med grafiske kart og søkemekanismer [Con87, Kah01, Bru99]. Grafiske kart kommer i mange former, men alle strever etter å gi brukeren en følelse av posisjonen sin og strukturen på nettverket.

Navigeringstjenestene har som formål å hjelpe brukeren med å svare på disse tre spørsmålene [Nie99]:

- Hvor er jeg?
- Hvor kan jeg gå?
- Hvor har jeg vært?

Vår motivasjon for utvikling av Lupen er at grafiske kart på et nettsted kan øke brukerens forståelse av sin posisjon i nettverket, og gi et raskt svar på disse tre spørsmålene. Disse spørsmålene står svært sentralt i navigeringen, og vil utgjøre basisen for hvordan vi vurderer andre nettapplikasjoner i kapittel 3 og i vår egen prototype:

Brukeren må se sin nåværende posisjon, tidligere besøkt noder og potensielle nye ruter.

2.8.3 Brukernes vaner

Det er vanskeligere å lese tekst på en dataskjerm enn på papir. Det ser også ut til at opplevelsen av å være på Internett gjør at brukeren har en viss utålmodighet. En undersøkelse av Nielsen [Nie99] viste at brukeren ikke leser teksten som en strøm av ord. Brukeren *skanner* teksten, og velger ut nøkkelord, setninger og paragrafer. Tekst som ikke er av interesse ser ut til å bli ignorert. 76 % av brukerne skannet enhver side de kom over. Grunnene til dette kan være mange:

- Internett er et brukersentrert medium, og kanskje brukeren føler at de må bevege seg rundt og klikke på ting.
- Siden Internett er så stort har brukeren ingen ide om den besøkte siden er den de egentlig ønsker, og vil derfor ikke investere for mye tid på hver side.
- Et moderne menneske har lite tid, og vil ikke arbeide for hardt for å finne informasjon [Nie99]

For å fange brukerens interesse er det derfor viktig at forfatter skriver teksten på en kort og konsis måte. Tittelen på siden bør være kort og informativ. Det samme gjelder for overskrifter. Erfaringer tilsier også at brukerne foretrekker korte sider som raskt lastes ned fremfor større sider med mer innhold [Nie99].

Vi vil også hevde at de samme aspektene gjelder nettkartene. Store og kompliserte nettkart vil ikke fange interessen til en bruker som ikke vil bruke tid på å sette seg inn i detaljer. For at nettkartene skal være brukbare må de være intuitive og enkle å bruke.

Vi tar med oss disse aspektene videre når vi skal se på noen eksempler på grafiske kart i Kapittel 3: *Kartene må være enkle, intuitive og raske.*

2.8.4 Lokal og global sammenheng

Ihlstrøm[Ihl99] refererer til Thuring[Thu95] som mener det er to faktorer som er viktige for å forstå et hyperdokument: Som en positiv faktor kan man bruke forståelse, og som en negativ faktor kan man bruke kognitiv overbelastning. Forståelse er delt i lokal og global sammenheng. Lokal sammenheng gjelder forståelsen av sammenhengen mellom setninger og paragrafer. Global sammenheng er den forståelsen som man får ut i fra kapitler, store bolker og flere dokumenter. For et hypermediesystem betyr det at man kan skille mellom nivået inni en node eller et dokument, til nivået mellom flere noder, som for eksempel på WWW. For å øke den globale sammenhengen må forfatterne gi brukerne en plattform hvor de kan forstå hvordan nodene er forbundet til hverandre.

Nielsen viser at brukerens posisjon i et nettsted må vises på to forskjellige nivåer: Relativt til Internett, og relativt til nettstedets struktur. Posisjonen til nettstedet i forhold til Internett er vanskelig å vise siden Internett ikke har noen fastlagt struktur. Men man

kan i alle fall sørge for at alle dokumentene i nettstedet har en konsis identifikator som utvetydig viser at man befinner seg på det aktuelle nettstedet. En god måte å gjøre dette på er å vise landemerker på ethvert dokument (som for eksempel en logo som også tar brukeren til startsiden). [Nie99]

Begge disse to aspektene er viktige for å svare på spørsmålene om hvor brukeren er, hvor han kan gå, og hvor han har vært. Ved å se den globale sammenhengen kan brukeren finne ut hvor mye av hypermediebasen han har besøkt, og hvor mye som gjenstår. Den lokale sammenhengen gir et mer detaljert bilde av hva brukeren faktisk har opplevd og lest av informasjonen.

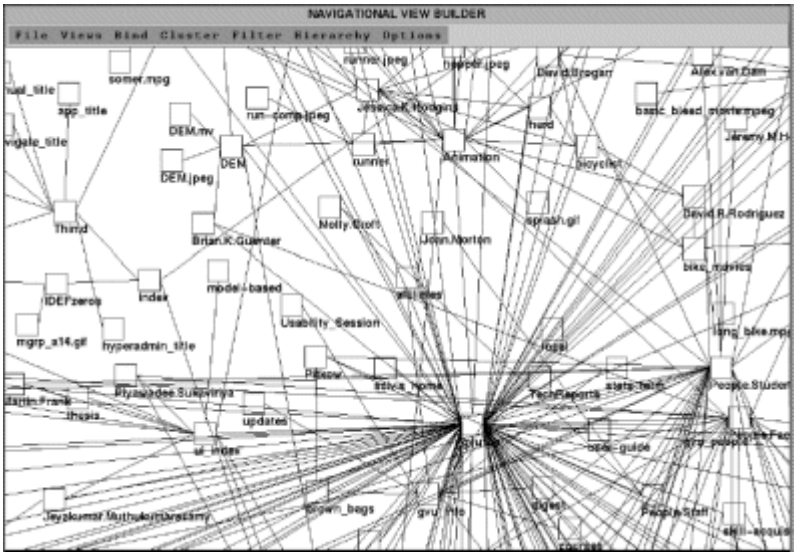
Brukerens posisjon må sees i både global og lokal sammenheng.

3 Teori og eksempler på nettkart

3.1 Innledning

Nettkart er et av de beste verktøyene for å navigere og orientere seg i et hypermediesystem. Det kan vise brukerens nåværende posisjon på nettstedet, hvilke sider han tidligere har besøkt, og mulige fremtidige besøk. I tilegg gjør nettkartene det enkelt å besøke sider som strukturelt ligger langt fra hverandre [Ihl99]: I stedet for klikke på flere lenker for å komme fram til destinasjonen, er et enkelt klikk på en node nok. Men å konstruere et effektivt nettkart er en meget vanskelig oppgave. Vi vil nevne spesielt utfordrende problemer:

- Nettkart vises i layouter på enten 2 eller 3 dimensjoner, selv om nettverkene generelt har flere dimensjoner.
- Selv om en layout kan vise 2 eller 3 dimensjoner, vil strukturen være meget kompleks for alle hypermediesystem som har en viss størrelse.
- Når størrelsen på nettverket øker, blir det vanskelig å få plass til alle denne informasjonen på skjermen. Hvis størrelsen blir redusert blir detaljene for små til å sees. Det blir en konstant kamp om plassen.
- Å kun vise strukturen er ikke nok. Brukeren må også få en ide om innholdet i nodene og lenkene [Muk].



Figur 3-1 Et eksempel på en dårlig struktur i et nettkart.

Figur 3-1 viser et av nettstedene til Georgia Institute of Technology, sett igjennom nettkartet Navigational View Builder. Her ser vi et sammensurium av noder og lenker. Det er umulig å forstå nettstedets struktur, og man får ingen annen informasjon om nodene unntatt tittelen. Det er vanskelig å se sin egen posisjon, og hvilke sider som lenkes til fra den. Ingen informasjon gis om hvilke noder man har besøkt. Vi kan heller ikke se omfanget av nettstedet, da skjermbildet ikke er stort nok. Til tross for alle disse åpenbare manglene, gir faktisk nettkartet et sannferdig bilde av den multidimensjonale strukturen til nettstedet. (Det må nevnes at Navigational View Builder ikke bruker denne strukturen som til vanlig, men brukte det som et eksempel på dårlig strukturering.)

Med disse problemene i tankene vil vi i dette kapittelet vise hvilke metoder og funksjoner som man kan bruke for å få et nyttig og funksjonelt nettkart. Det finnes styrker og svakheter i alle sammen. Vi vil ikke hevde at dette utvalget verken er representativt, eller et gjennomsnitt. De er valgt ut i fra hva man kan lære og har lært av dem. Denne lærdommen vil vi eksplisitt markere, og vise hvordan vi har brukt den i utviklingen av Lupen.

Fordelen med de fleste nettkart er at de gir en forståelse av hva nettstedet inneholder, og brukeren kan oppdage ressurser som han tidligere ikke var klar over eksisterte. Å finne omfanget av nettstedet vil være lettere. Særlig studenter som følger kurs eller leser pensum på Internett vil kunne få en ide over mengden materiale, og tilpasse innsatsen deretter. Brukere får det også lettere med finne ressurser ved å gjenkjenne ting visuelt, istedenfor å måtte huske navn. Et av de største problemene er mangel på plass. Jo mer detaljer og informasjon man viser, desto større er risikoen for at nettkartet blir uoversiktlig. Et godt nettkart har en god balanse mellom disse to tingene.

3.2 Statiske nettkart

3.2.1 Funksjonalitet

Det finnes to typer nettkart: dynamiske og statiske. Dynamiske nettkart oppdateres automatisk når det skjer en forandring i hypermediebasen, mens statiske nettkart må redigeres av forfatter.

Vi vil ikke vurdere de statiske nettkartene ut i fra lærdommen fra Kapittel 2, da statiske nettkart ikke er interaktive. Vi tar dem kun med for å vise spennet i design, noe som også gjelder dynamiske nettkart.

Statiske nettkart begynte å bli vanlige fra 1996, da Internett virkelig begynte å vokse i omfang. De har to hovedfunksjoner:

- Gi en oversikt over nettstedets innhold
- Gi rask tilgang til de forskjellige sidene

Nielsen mener brukeren ved et raskt blikk på nettkartet skal få en god oversikt: Hvordan nettstedet er bygd opp, hva det inneholder av informasjon, og hvor de kan gå. Får å gjøre dette mulig er det viktig at nettkartet ikke er så stort at det ikke får plass på skjermen [Nie02]. Rask tilgang til sidene betyr at brukeren kan hoppe direkte og dypt inn i nettverket, uten å gå omveier via flere andre sider.

Clemens mener de statiske nettkartene kun er brukbare hvis nettstedet ikke er meget lite eller svært stort. Dette er fordi det ikke blir plass til nok informasjon på nettkartet [Cle03].

Kahn deler de statiske nettkartene inn i fire grupper, alt ettersom hvilken visuell projeksjon de benytter: **Hierarkiske lister** som er organisert vertikalt og/eller horisontalt, **sirkelbaserte projeksjoner**, **progressive avsløring** og **metaforer**. Det er mulig å bruke en kombinasjon av disse projeksjonene samtidlig. I tilegg er det vanlig å bruke forskjellige visuelle variabler som: Fontstørrelse, farger, innrykk, grad av nærhet, linjer, symboler og ikoner [Kah01].

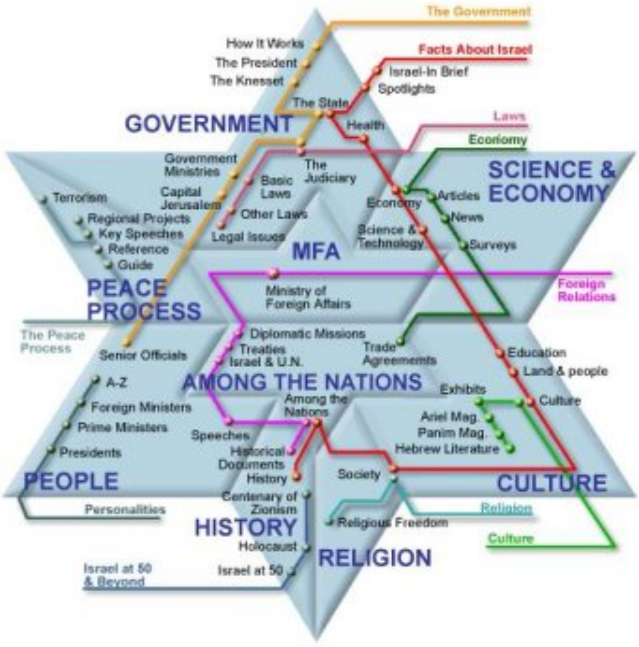
3.2.2 Presentasjon

Figur 3-2 Det gamle nettkartet til Apple.

I Figur 3-2 ser vi Apples gamle nettkart. Det er laget av en HTML-tabell som inneholder mange små bilder. Samlet ser alle de små bildene ut som et stort bilde. Bak hvert lille bilde (kategori) ligger ankeret til lenken. t. Vi kan se alle de syv hovedkategoriene markert med egne, klare farger. Under hver av disse hovedkategoriene blir det listet opp noen underavdelinger. I midten av sirkelen vises logoen til Apple, og alle hovedkategoriene stråler ut i fra den. Nettkartet er behagelig og friskt å se på.

Selv om man får en ide over det konseptuelle omfanget til Apples nettsted ved å se på antallet hovedkategorier, gis det lite detaljert informasjon. Hvor mange sider eksisterer i hver underkategori? Er det en side eller hundrevis? Vi får ingen informasjon om strukturen, hvorfor er for eksempel kategorien ”Technology & Research” plassert øst for logoen? Ved klikk på en av kategoriene hjelpes man bare på en del av veien. Man må stole på at nye lenker vil hjelpe deg videre derifra. Dette bryter med nettkartenes funksjon med å gi rask tilgang til dype dokumenter.

Hvis man skulle utvide Apples nettkart med flere detaljer eller flere hovedkategorier, må forfatter omorganisere hele nettkartet.



Figur 3-3 Nettkartet til Israels Utenriksdepartement[21]

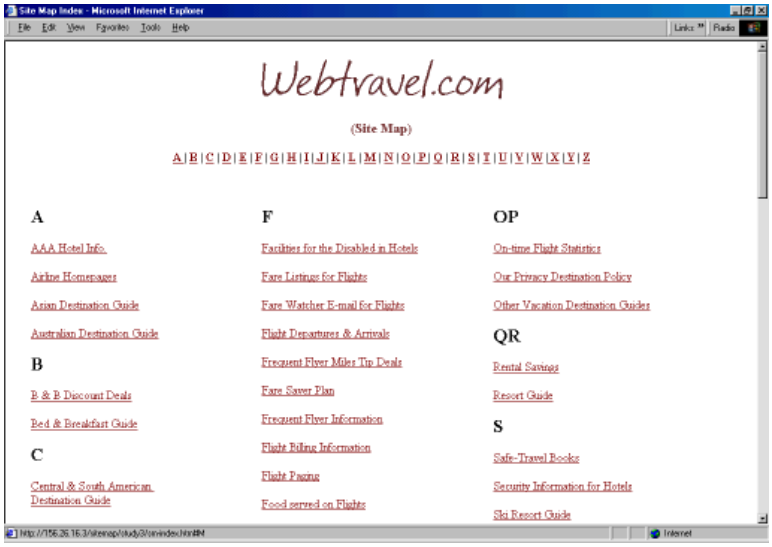
I Figur 3-3 ser vi et nettkart som bruker en metafor. Det bruker metaforen fra Davidsstjernen som er Israels nasjonalsymbol. Taggene til stjernen indikerer forskjellige kontekster. Linjene som går på kryss og tvers indikerer emnene som man igjen kan se i de forskjellige kontekstene. Linjene bruker i tillegg en metafor for kart over tunnelbaner. Det er meget pent og godt organisert. Ulempen er at en oppdatering av dette nettkartet ikke er lett. Hvis man ville lagt til konteksten ”idrett” måtte man forkaste hele designet.

Kahn og Nielsen hevder at metaforer ikke bør brukes i det hele tatt da det er meget vanskelig å finne en metafor som fungerer. De er vanskelige å oppdatere og i tillegg vil ikke brukerne sette seg inn i en ny måte å tenke på, bare for å bruke nettkartet [Nie01, Kah01].

[21] Israels Utenriksdepartement, http://www.mfa.gov.il/mfa

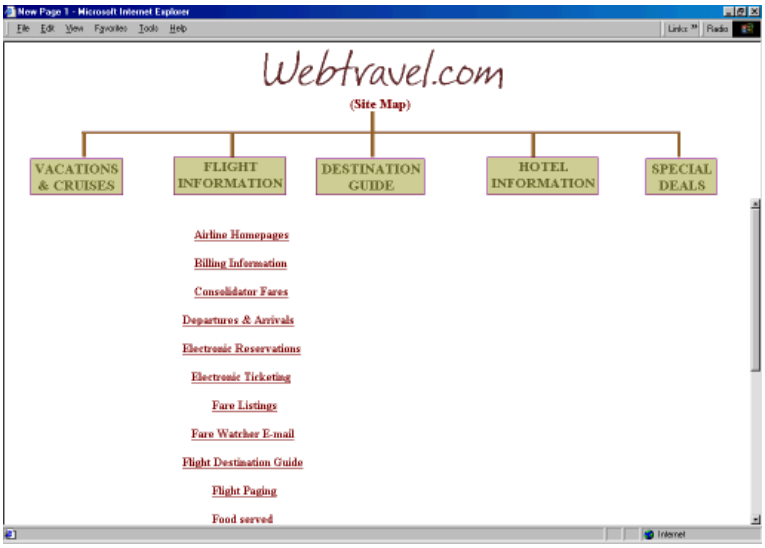
3.2.3 Brukervennlighet

Siden statiske nettkart er skrevet direkte av forfatter, eksisterer det egentlig ingen begrensninger. Det er kun forfatters fantasi som setter grenser. Men erfaringer viser at brukerne foretrekker kjente løsninger. Det er viktig å huske på at brukerne benytter nettkart når de allerede har gått seg vill. Da vil man ikke forvirre dem ytterligere ved å kreve at de setter seg inn i et nytt system. Nielsen går så langt som å hevde at brukerne *hater* løsninger som ikke bruker allerede kjente standarder [Ber00, Nie02].



Figur 3-4 Nettkart som bruker alfabetisk liste.

Noen nettkart benytter seg av alfabetiske lister. I Figur 3-4 ser vi et eksempel. Dette er en standard som vi alle er kjent med fra bøker og andre indekser. Slike lister kan bli svært lange, og man trenger vanligvis å bruke rullemenyen for å se hele listen. En annen ulempe er at brukeren ikke vet hvilket ord som blir brukt: Hvis man leter etter informasjon om ”yoghurt” er den kanskje registrert i listen som ”meieriprodukter”. Undersøkelser har vist at brukeren unngår slike lister til fordel for lister som viser innholdet oppdelt i kategorier [Ber99].



Figur 3-5 Et nettkart som bruker progressiv avsløring.

I Figur 3-2 så vi et eksempel på et nettkart som er oppdelt i kategorier. Men hvis antallet kategorier blir stort, vil man få et plassproblem. Progressiv avsløring kan være til hjelp i slike situasjoner, da kun en del av innholdet blir vist. I Figur 3-5 ser vi et slikt eksempel. Brukeren må trykke på knappen ”Flight Information” for å se alle underkategorier. Bernard fant ut at brukerne foretrakk progressiv avsløring framfor alfabetiske lister. Men de foretrakk også en fullstendig visning framfor en progressiv avsløring [Ber99].

Nielsen kritiserer progressiv avsløring da man ikke ser omfanget og dybden med en gang. I tillegg gir det heller ikke rask adgang. Igjen kan brukeren risikere å måtte klikke seg meget dypt ned før han finner fram til det han leter etter [Nie02].

3.3 Tredimensjonale nettkart

Noen nettkart bruker benytter tredimensjonal grafikk for å vise innholdet i et nettsted. Dette er generelt en dårlig ide. Selv om man kan benytte mange teknikker og teknologier, vil resultatet sjelden bli bra. Det finnes mange fantastiske visuelle eksempler, men de fungerer bare ikke på en funksjonell måte:

- Det er lenge siden brukerne ble imponert av 3D, så det blir ikke ”nytt og spennende”.

- Skjermen og musen er todimensjonale medier, og kan ikke gi brukeren en reell, tredimensjonal opplevelse uten ekstrautstyr som 3D-hjelmer.
- Det er vanskelig å kontrollere navigasjonen i 3D, og dette forhindrer rask tilgang.
- 3D nettkart benytter vanligvis spesiell programvare som må installeres på klientmaskinen. Dette burde ikke være nødvendig bare for å se et enkelt nettkart.
- Mennesker er mer vant å navigere i informasjonsmedier som er todimensjonale. [Nie99]

Hvis vi summerer opp disse argumentene mot 3D blir resultatet kognitiv overbelastning og desorientering, nøyaktig hva nettkartene skal forhindre. Som et resultat av dette vil vi ikke gå nærmere inn på bruk av 3D i nettkart.

3.4 Dynamiske nettkart: Teori

Til forskjell fra statiske nettkart som blir lagd av forfatter, blir dynamiske nettkart lagd automatisk av et program. Dette programmet henter ut visse data fra hypermediebasen og konstruerer ut i fra disse et kart i henhold til en matematisk formel.

Det er vanskelig å finne en enhetlig definisjon på dynamiske nettkart, men de fleste deler følgende funksjonalitet:

- Nettkartet kan forandres ut i fra brukerens handlinger
- De benytter en matematisk logikk for å vise nodene på skjermen
- De finner automatisk sine data fra hypermediebasen

Kahn stiller to kjernespørsmål til de dynamiske nettkartene:

- Kan en objektiv beskrivelse av et nettsted være nok til å lage et kart? Kan vi lage et kart ut i fra data som automatisk blir hentet i hypermediebasen? Slike data er som regel dokumentenes HTML-kode, titler, størrelse, metadata og så videre.
- Hvordan skal vi visualisere disse dataene på en måte som er funksjonell?

Disse spørsmålene må besvares i hvert enkelt tilfelle.

3.4.1 Klient/Tjener forhold i nettleser

Tjenere er som regel store datamaskiner med mye mer datakraft enn personlige datamaskiner. En **webtjener** er i praksis et program som er installert på en tjener tilknyttet Internett. Dette programmet holder styr på de ressurser som ligger på maskinen, for eksempel hjemmesider. Webtjener svarer på ønsker fra nettlesere (**klienter**) om overføring av nettfiler (HTML, bilder og eventuelt andre ting som tilhører nettsiden). Begrepene "tjenerside" og "klientside" refererer til om det er serveren eller om det er klienten som utfører prosessen. Webtjeneren tar imot ønsker fra nettleseren, når f.eks. brukeren klikker på en lenke. Server og nettlesere kommuniserer via http-protokollen. Ved vanlig bruk av nettleser kan man kun lese innholdet på tjeneren. Sletting og modifisering er ikke tillatt.

Forholdet mellom klienter og tjenere i nettleseren gjør det vanskelig å lage et dynamisk nettkart som reagerer i forhold til brukerens handlinger i nettleseren. Nettleseren etterspør bare et dokument fra webtjeneren, som så leverer dette. Webtjeneren gir ingen ekstra kommandoer til klientens nettkart om at et dokument nettopp er levert.
Så når brukeren klikker på en lenke i nettleseren, vet ikke nettkartet at dette skjer. Følgelig kan ikke nettkartet bygge opp en historie over besøkte dokumenter, eller på en annen måte vise at et nytt dokument ble levert. Den eneste måten nettkartet kan få beskjed om en ny side, er hvis brukeren klikker på en node/lenke i selve nettkartet. Nettkartet er med andre ord **tilstandsløst.**

For å omgå dette problemet må man lage en egen tjeneste på webtjeneren som gir ekstra kommandoer til nettkartet ved etterspørsel av et dokument (Webtjenere gir ikke disse kommandoene, da nettleserne ikke har noen dynamiske nettkart som standard).

Hvis man vil unngå tilstandsløse nettkart, må man som sagt utvikle en egen tjeneste som støtter dette på tjeneren. Dette gjør det mulig å ha avanserte funksjoner i nettkartet, men

det vil kun gjelde på akkurat det nettstedet. Vårt eget nettkart Lupen er ikke tilstandsløs. Vi skal se hva dette innebærer senere i oppgaven.

Tilstandsløs eller ikke, mange nettkart bruker **Java** som plattform. Brukerne kan laste inn egne Java-programmer på klientsiden kalt **appleter,** som kan kjøres i nettleseren. Dette er en meget sikker måte å kjøre programmer på, da Java appleter ikke har skrivetilgang til klientens harddisk. Java er i tilegg plattformsuavhengig, det vil si at samme applet kan kjøres på mange forskjellige operativsystemer.

3.4.2 Edderkopper

Som vi har vært inne på i 2.4 er et nett en abstraksjon av lenkene som finnes mellom hyperdokumentene i hypermediebasen. For å visualisere dette nettverket må man trekke ut relevant informasjon og organisere det på en eller annen måte.

Denne oppgaven blir vanligvis utført av en **edderkopp** (engelsk: Crawler/spider). Edderkoppen blir gitt en startadresse (vanligvis toppdokumentet på nettstedet), og traverserer så nettverket i tur og orden. Mens den gjør dette bygger den opp en **indeks** over all relevant informasjon. Denne informasjonen er for eksempel filstørrelse, antall lenker i hvert dokument, og eventuell metadata. Edderkoppen vil som regel begrense seg til et enkelt nettsted eller **domene**[22]. Dette er fordi traversering av hele Internett vil ta alt for lang tid, og kreve enorm datakraft. Vanligvis bruker kun store søkemotorer (eks. Google og Yahoo) edderkopper som traverserer hele Internett.

Edderkoppen må hele tiden vurdere om en lenke fører utenfor de grensene den blir gitt. Hvis den finner en slik lenke må den avgjøre om den skal ignorere den, eller hente informasjon om måldokumentet. I tillegg må den huske hvilke lenker den allerede har besøkt for å unngå duplikat informasjon [Mil98].

Edderkoppen kan fungere både på tjener og klientsiden. Edderkopper på tjenersiden har som regel ingen annen funksjon enn å oppdatere nettsteder eller å prøve å holde en

[22] Et navn som identifiserer en organisasjon eller lignende på Internett. For eksempel er adressen www.ntnu.no er domenenavn hvor ”no” er toppnivådomenet som identifiserer at webtjeneren befinner seg i Norge, ”ntnu” er domenet som er registrert av NTNU, mens ”www” identifiserer en bestemt tjener under ntnu-domenet. Domenenavnet er knyttet til en IP-adresse, som lokaliseres av en DNS-tjener.

hypermediebase konsis mellom flere brukere. Edderkopper på tjenersiden har som regel full tilgang til alle dokumenter på nett-tjeneren. En edderkopp som kjører på klientsiden har bare tilgang til dokumenter som har normal nett-tilgang, det vil si vanlig lesetilgang.

Det er flere ting som kan gå galt ved traverseringen og innhentingen av informasjon: Webtjeneren kan være nede og dokumenter kan være flyttet eller slettet. Dokumenter og lenker kan være i formater som edderkoppen ikke gjenkjenner (eksempel: Flash, Java, CGI, Paint Shop Pro psp-filer og så videre). I tillegg kan deler av nettverket være passordbeskyttet, eller kun vise dokumenter avhengig av eksplisitt input [Kah01].

3.4.3 Layout

For at et dynamisk nettkart skal være brukbart må brukeren kunne se en logikk bak plasseringen av nodene. I Figur 3-1 kunne vi ikke se noen logikk, og resultatet var et uoversiktlig rot. Med andre ord trenger det dynamiske nettkartet en layout. Det er for mange typer layouter til at vi kan kategorisere dem, eller vise alle.

Layout balanserer mellom to overordnede ting. For det første må den gi brukeren en kognitiv forståelse av nettverkets struktur. For det andre må den være mulig å lage automatisk ut i fra de data som er tilgjengelig.

I tillegg finnes det flere vanskelige aspekter:

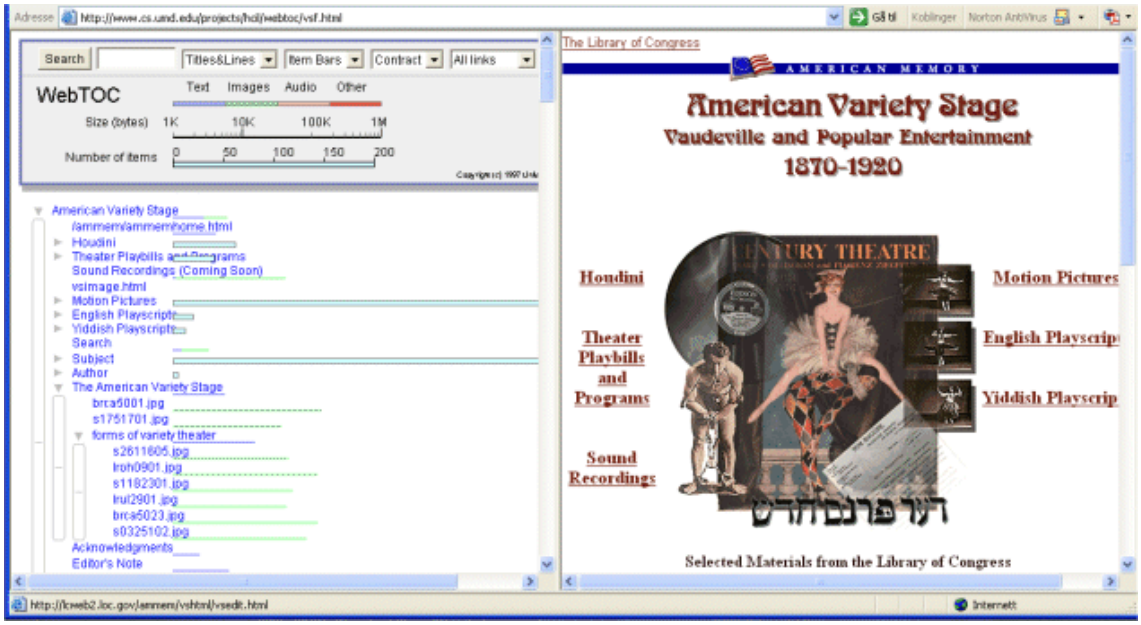
- Skjermen har begrenset med plass. Paradokset er at de nettstedene som mest trenger nettkart for navigering, også som regel har mange noder og lenker.
- Layouten kan også avhenge av selve innholdet i dokumentene. Dokumenter kan bli rangert i henhold til likhet, nærhet og så videre. Dette kan være vanskelig å vise visuelt.
- Utviklingen av matematikken bak layout kan være ressurskrevende og vanskelig.
- Layouten kan måtte ta hensyn til intern struktur i hypermediebasen, for eksempel katalogstrukturen.

- Layouten bør støtte animering av nodene og lenkene. Brukeren bør se overgangen fra en situasjon i nettkartet til en annen, hvis ikke kan han miste orienteringen.

Kahn påpeker at de fleste dynamiske nettkart viser en eller annen form for hierarki, og at de kartene som ikke gjør det heller ikke prøver å vise en større del av nettverket. [Kah01]. Vi har samme erfaring. Det er naturligvis vanskelig for brukeren å få en konseptuel forståelse hvis man fjerner det hierarkiske aspektet i layouten. For å se hvilken node han besøker må denne noden vises på en ”viktigere måte” enn de andre. Vår erfaring er også at de mest brukbare dynamiske nettkartene viser hierarkiet igjennom en sirkelbasert layout (Mapuccino, Inxight Star Three).

Layout kan naturligvis benytte seg av de samme visuelle variabler som de statiske nettkartene (fontstørrelse, farger, koner osv). Siden dynamiske nettkart er egne programmer kan man i tilegg lage egne variabler. Java appleter tillater for eksempel visning av verktøytips. Vi skal komme nærmere inn på slike ting i neste kapittel.

3.5 WebTOC



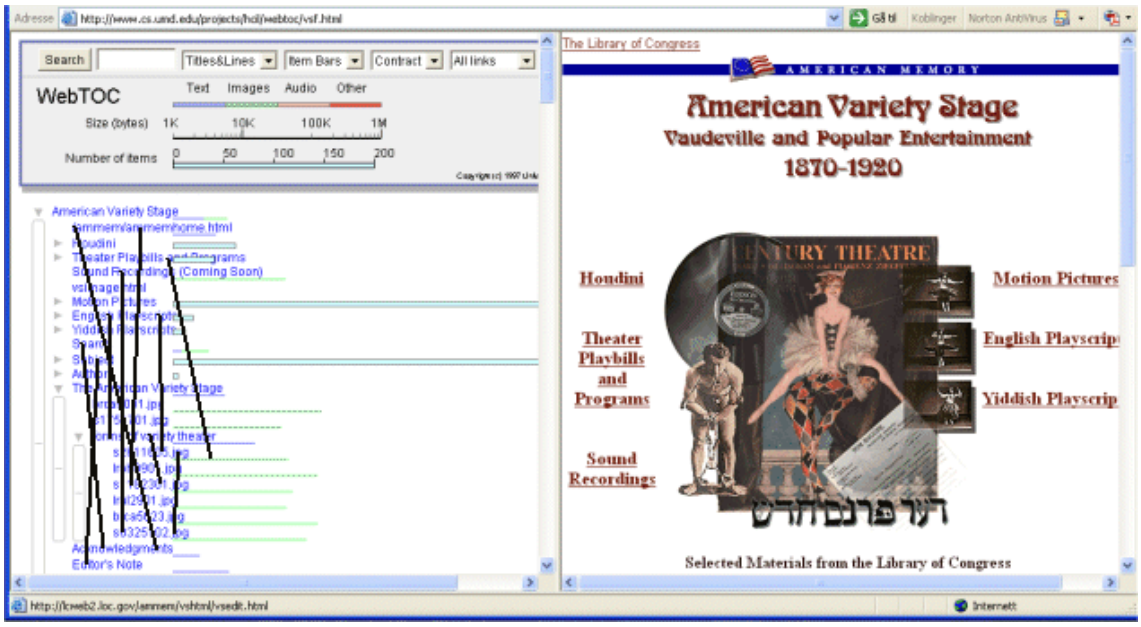
Figur 3-6 WebTOC brukt på Library of Congress i USA

WebTOC er et dynamisk nettkart som viser en innholdsliste i en ramme. Det er utviklet av University of Maryland. Programmet er skrevet i Java, og gir en utvidet funksjonalitet til en ellers vanlig innholdsliste. Det består av to deler: en edderkopp og en visuell del. Begge disse delene befinner seg på klientsiden. Ved start blir edderkoppen gitt en URL som den begynner å traversere ut i fra. Edderkoppen bygger opp en indeks som den visuelle delen så kan generere en hierarkisk representasjon fra. I Figur 3-6 ser vi WebTOC bli brukt på et av nettstedene til Library of Congress i USA. I dette eksemplet søkte edderkoppen igjennom 3400 dokumenter, noe som tok omkring 4 sekunder.

Selve layouten følger en tremodell. En tremodell har en toppnode som alle andre noder kan nåes i fra. Under toppnoden blir neste nivå noder vist, under dem igjen neste nivå og så videre. En tremodell er identisk til hvordan Microsoft Utforsker viser katalogstrukturen i Windows. Treet kan ekspanderes og kollapses, det vil si det viser en **progressiv avsløring**. Ved trykk på en node, vises det gjeldende dokumentet i hovedrammen. Hver dokumentnode viser en farget linje som indikerer størrelsen på filen. Fargen på linjen viser hvilke typer media som filen inneholder (video, bilde, tekst eller lyd). Hvis dokumentet inneholder lenker til andre dokumenter, kan dette bli vist i en annen linje hvor lengden avhenger av antall lenker. WebTOC har også en innebygd søkemotor.

Kartet er intuitivt og enkelt å forstå. Brukeren trenger ikke mer forklaring enn det som allerede er synlig i bildet.

Layouten er spesielt egnet for en hierarkisk informasjonsstruktur. Det blir mer problematisk når informasjonsstrukturen er nettverksbasert, som på Internett. Det grunnleggende problemet blir da å tvinge en 3-dimensjonal struktur inn i en 2-dimensjonal struktur. Dette problemet har også matrise og hierarkisk nettverksstruktur. Det er vanskelig å vise refererende lenker som går mellom nivåene. Hvis man skal tegne en strek mellom slike noder, vil layouten potensielt bli full av mer eller mindre vertikale linjer. Vi kan se dette problemet i Figur 3-7, hvor vi selv har tegnet inn noen refererende lenker.



Figur 3-7 WebTOC med lenker mellom nodene (tegnet av oss)

WebTOC bruker også altfor stor plass på skjermen. Man burde i alle fall redusert plassen til bildet som viser hvilke farger som tilsvarer filtypene. Titlene på sidene som nodene viser burde også automatisk blitt kortet ned. Plassproblemet kunne delvis blitt løst ved å åpne WebTOC i et eget vindu, og gi en lenke til dette vinduet i dokumentrammen. WebTOC ville hele tiden vært tilgjengelig, men ute av syne.

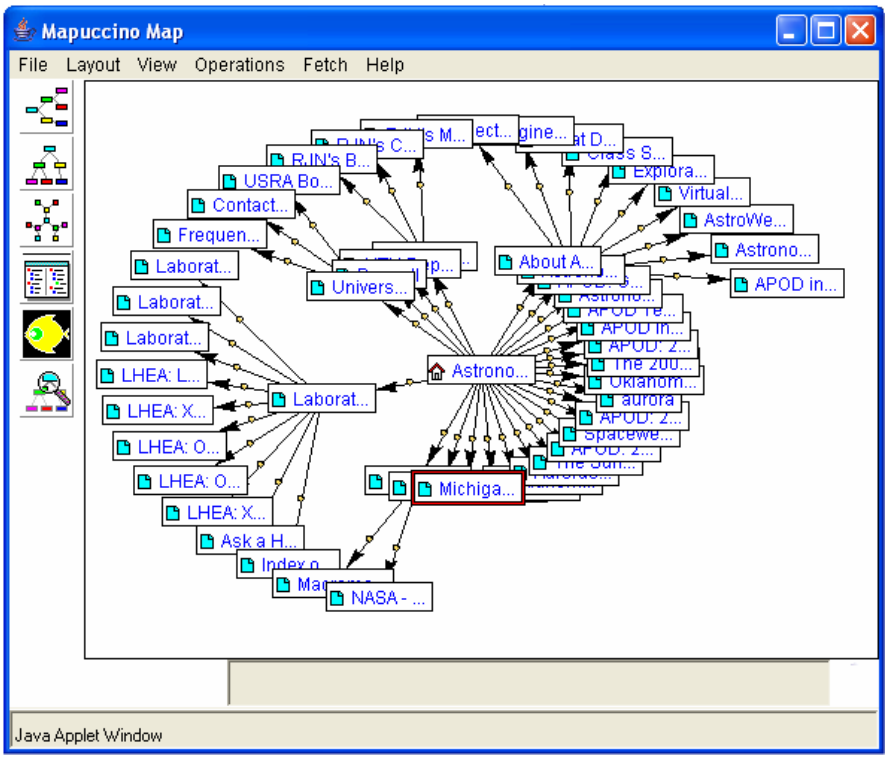
Programmet er tilstandsløst. Hvis brukeren har besøkt en side ved hjelp av klikk på lenker i selve nettleseren, vil ikke WebTOC gjenspeile dette.

Nodene forandrer heller ikke farge hvis man klikker på den, og dermed får man ikke noen følelse av hvor man er, eller har vært, i treet. Selv om WebTOC er tilstandsløst burde denne funksjonaliteten vært implementert. Treet blir også fort for langt for skjermen langt hvis man ekspanderer det, og dette hjelper ikke på situasjonen. Dette er spesielt et problem på en så stor nettside som Library of Congress. Brunk hevder dette gjør at slike layouter kun er brukbare til små og middel store nettsteder [Bru99].

Lærdom:

- Åpning av nettkartet i et eget vindu gir mer tilgjengelig plass.
- Ikoner bør forandre farge hvis de er besøkt.
- Nodene bør indikere forskjellige filtyper i dokumentet.

3.6 Mappucino

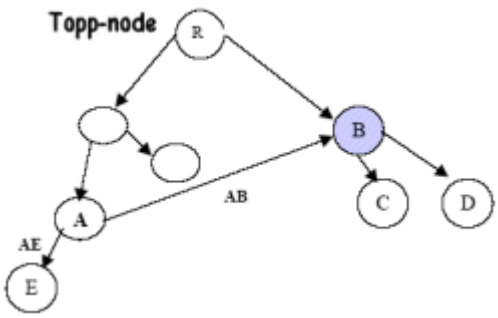


Figur 3-8 Mappucino brukt på APOD[23]. Sykliske lenker vises ikke.

Mappucino (tidligere WebCutter) er utviklet av IBM Haifa Research Lab i Israel. Programmet tilbyr 4 forskjelliga layouter: Hierarkisk, Tre, Sirkel og Fiskeøye. I Figur 3-8 ser vi et eksempel på vanlig sirkellayout. Avhengig av antall noder og hierarkisk forhold, blir nodene tildelt en viss prosentandel av sirklenes 360 grader. Disse nodene kan flyttes ved hjelp av musen.

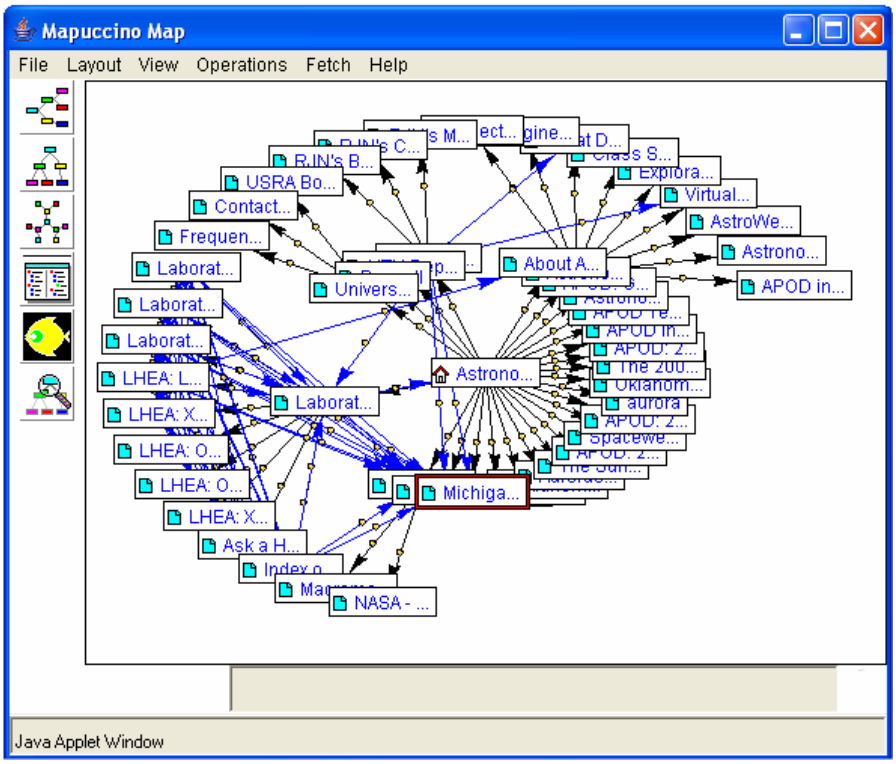
Mappucino skiller mellom to typer lenker: **sykliske lenker** og **ikke-sykliske lenker**. Ikke-sykliske lenker har et far til barn forhold: En far kan ha mange barn, men hvert barn kan bare ha en far. Dette skillet er naturligvis ikke naturlig i Internett, hvor alle noder kan lenke til alle andre noder. Skillet er kun en måte å prøve å tvinge alle noder inn i et todimensjonalt hierarki på selve layouten.

[23] Astronomy picture of the Day, http://antwerp.gsfc.nasa.gov/apod/



Figur 3-9 Forholdet mellom sykliske og ikke-sykliske lenker

I Figur 3-9 kan vi se dette forholdet. Lenker mellom R og B er en ikke-syklisk lenke. Lenken mellom A og B er en syklisk lenke. Ved å skille mellom visning av disse lenketypene vil antall streker på nettkartet gå ned. Dette gjør det lettere å se forholdet mellom nodene. Ulempen er naturligvis at brukeren ikke er klar over dette, og at det faktisk ikke gjenspeiler det reelle forholdet. Ved bruk av Mappucino kan brukeren selv velge om han vil se kun sykliske lenker, eller alle lenker [Hao99]. I Figur 3-10 ser samme situasjon som i Figur 3-8, men denne gangen vises de sykliske lenkene.



Figur 3-10 Mappucino med visning av sykliske lenker

Nodene kan vises med delvis eller full tittel. Tittelvisning og layout bestemmes av bruker. Mappucino kan også vise brutte lenker, bilder, og noder som befinner seg utenfor nettstedet. Dette indikeres av forskjellige, bitte små ikoner i nodene.

Edderkoppen kjøres av Webmaster som må lagre den resulterende indeksen som en egen HTML-fil. Filen inneholder alle parametere som Mappucino trenger for å konstruere nettverket som noder og lenker. Forfatter eller webmaster legger så ut en vanlig hypertekstlenke til denne filen. Når brukeren klikker på lenken, vil klienten starte opp Mappucino som en applet. Mappucino vil da også lese inn filen med parametrene, og ut i fra den konstruere layouten.

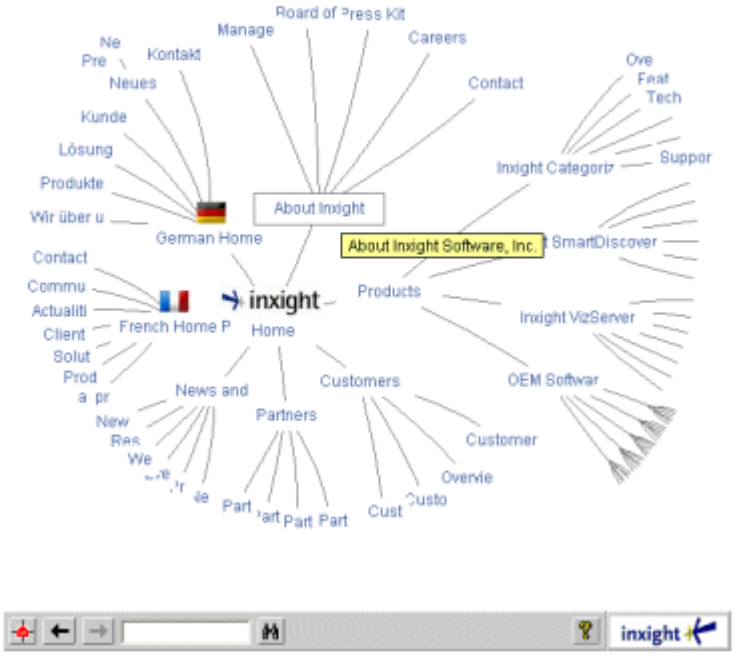
Mappucino åpnes som et eget vindu. Hvis programmet skal vise brutte lenker, bilder og eksterne noder, må dette spesifiseres av forfatter/webmaster ved kjøring av edderkoppen.

Layoutene er svært oversiktlige, og viser det innbyrdes forholdet på en god måte. Valget mellom sykliske og ikke-sykliske lenker letter rotet på skjermbildet. At man kan flytte på nodene er bra, særlig da de kan ligge delvis over hverandre. På samme måte som ved WebTOC vil visning av full tittel bli uoversiktlig, særlig hvis titlene er av svært ulik lengde. Mappucino indikerer heller ikke hvor brukeren er, da det er tilstandsløst. Noder som er eksterne kan som sagt vises, men dette indikeres ikke eksplisitt på nodene. De burde hatt annerledes farge, eller er ikon som indikerte dette.

Lærdom:

- Bør gi brukeren et valg mellom sykliske og ikke-sykliske lenker.
- Nodene bør kunne flyttes på.
- Nodene bør indikere om de er utenfor nettstedet.

3.7 Star Three



Figur 3-11 Eksempel på Star Three tatt fra hjemmesiden til Inxights[24]

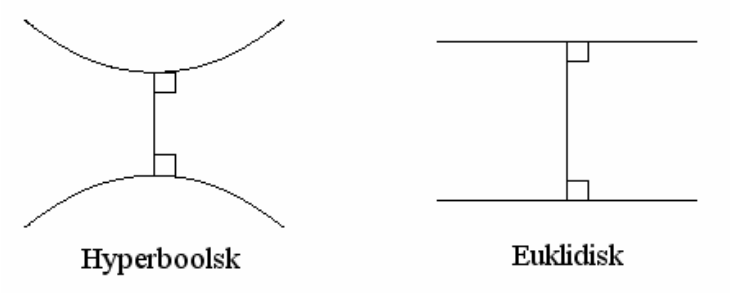
Inxight Software har utviklet et dynamisk nettkart ”Star Three”[25]. Det kjøres som en applet på klientsiden, men webmaster må spesifikt installere et program på tjeneren for å kunne støtte det. Star Three viser ingen annen informasjon om nodene enn tittelen på siden. Som WebTOC har den en innebygd søkemotor. I tilegg bruker Star Three verktøytips. Som vi kan se i Figur 3-11 vil den fulle tittelen på siden vises når musen beveger seg over ikonet.

Det er to ting som gjør dette programmet interessant. Den ene er at den benytter en hyperboolsk layout. Det andre at den har en intern historie.

Hyperboolsk layout betyr at man benytter en hyperboolsk geometri. I vanlig euklidsk geometri vil to linjer som er vinkelrette mot en tredje linje aldri møtes. De vil alltid ha den samme avstanden fordi de alltid vil være parallelle.. I hyperboolsk geometri vil

[24] http://www.inxight.com/map
[25] Produktet er kun kommersielt tilgjengelig i programpakken ”Inxight VizServer”. Siden vi ikke har kjøpt den, har vi svært begrenset informasjon om appleten.

linjene bøyes av fra hverandre med økende avstand i takt med avstanden til skjæringspunktet. Vi kan se dette i Figur 3-12:



Figur 3-12 Hyperboolsk og euklidsk geometri

Som vi kan se i Figur 3-11 er Star Three sirkelbasert som Mappucino. Men jo lengre vekk fra sentrum av sirkelen man ser, jo mer bøyes strekene, og jo flere noder vises. Nodene blir også progressivt mindre og mindre til de ikke er synlige som annet enn små prikker. Det er Star Three sin bruk av hyperboolsk geometri som gjør dette mulig. Brunk hevder hyperboolsk layout gjør det mulig å vise 10 ganger så mange noder på samme plass som en vanlig todimensjonal layout [Bru99].

Layouten er også animert, slik at brukeren kan ”trekke og dra i nodene” for å sette dem i fokus. Layouten fungerer som en ball, hvor man kan rotere rundt på ballen for å sette fokus på forskjellige områder. Dette er menget enkelt, og dette gjør at brukeren ikke mister orienteringen. Den røde knappen i nederste, venstre hjørne stiller fokus inn på startsiden igjen.

Star Three implementering av en historie, gjør at brukerne kan navigere seg frem og tilbake i historien ved hjelp av pilknappene i nederste, venstre hjørne. Historien er kun for den tiden (sesjonen) brukeren ser på kartet. Hvis han faktisk går til en side og så tilbake igjen, vil appleten lastes inn på nytt. Da blir historien nullstilt. Den er dermed delvis tilstandsløst. Men bruken av historien gjør at brukerne ikke behøver rulle rundt på bildet hele tiden, men kjapt kan få et overblikk av hvordan han kom til et sted.

Ulempen ved Star Three er mangelen på detaljert informasjon om nodene. Det burde vært mulig å se hvilke sider som har bilder, animasjoner og så videre. I eksemplet er dette ikke noe problem, fordi sidene har kun korte ord som ”Customer”, ”Products” og

så videre. Hvis man skulle brukt appleten på Library of Congress, hvor titlene var lange, ville det antakelig ikke fungert like bra.

Lærdom:

- Nettkartet bør ha en intern historie
- Layouten bør være animert
- Hyperboolsk layout kan vise flere noder

3.8 Hva har vi lært?

I Kapittel 2 viste vi at et dynamisk nettkart må oppfylle visse forutsetninger for at brukerne skal kunne få fullt utbytte av dem. Disse var at:

- Brukeren må se sin nåværende posisjon, tidligere besøkt noder og nye ruter.
- Kartene må være enkle, intuitive og raske
- Brukerens posisjon må sees i både global og lokal sammenheng.

Dette er de overordnede målene. For å nå disse målene har vi sett at forskjellige programmer benytter ulike arkitekturer og layouter. Vi skilte ut de viktigste av disse, og listet dem opp underveis. Denne lærdommen har vi aktivt brukt ved utviklingen av Lupen[26]:

- Nettkartet bør ha en intern historie
- Layout bør være animert
- Hyperboolsk layout kan vise flere noder
- Åpning av nettkartet i et eget vindu gir mer tilgjengelig plass.
- Ikoner bør forandre farge hvis de er besøkt.
- Nodene bør indikere forskjellige filtyper i dokumentet.
- Bør gi brukeren et valg mellom sykliske og ikke-sykliske lenker.
- Nodene bør kunne flyttes på.
- Det bør vises om en node er utenfor nettstedet/domenet.

[26] Vi sier lærdom til forskjell fra spesifikasjoner. Dett er fordi lærdommen kom underveis i utviklingen, og ikke som en spesifikasjon i forkant av utviklingen.

4 Opsys, det eksisterende system

4.1 Innledning

Dette kapittelet vil beskrive hvordan Opsys fungerer, og hvilke tjenester som tilbys.

Hypersystemet Opsys er utviklet på Institutt for Datateknikk og informasjonsvitenskap ved NTNU under ledelse av Arvid Staupe. Det har vært et utviklingsprosjekt siden 1995, og flere hovedfagstudenter har vært med. Det er en lokal løsning på et hypersystem. Det vil si at Opsys ligger på en tjener på IDI med begrenset brukertilgang.

Vi vil fra og med dette kapittelet vise flere eksempler på essensiell-Java kode. For lesbarhetens skyld har vi forenklet koden hvor det er mulig. Likevel forutsetter vi at leseren har en viss innsikt i grunnleggende programmering.

4.2 Oppstart av Opsys

Lenken til Opsys finnes på fagets hjemmesider. Den første siden man møter har et bilde av en dør, pluss tre små tester som sjekker at nettleseren er konfigurert ordentlig. Disse to første testene sjekker at brukeren har JavaScript og Java slått på i nettleseren. Den siste sjekker om Swing[27] er installert. Hvis noen av disse tre kriteriene ikke møtes, vil brukeren få beskjed om dette. Brukernavn og passord er påkrevd for å fortsette. Ved å klikke med musen på døren, kommer det et lite identifikasjonsvindu opp hvor man da taster dette inn. (Brukernavn og passord tildeles av øvingsleder ved semesterstart.

[27] Swing er en plug-in fra Sun Microsystems. Den gjør det mulig å kjøre appleter i nettleseren. I tillegg benytter den grafiske, egenkomponerte komponenter.

Figur 4-1 Skjermbildet til OPSYS

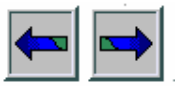
Den grensesnittet som brukeren først stifter bekjentskap med vises i Figur 4.1. Dette grensesnittet er delt i to rammer. Den øverste rammen er Menyrammen, og denne representerer alle tjenester systemet tilbyr. Menyen presenteres i form av ikoner som visuelt skal si noe om tjenesten som befinner seg bak referansen. Den nederste rammen er dokumentrammen. Her presenteres fagets dokumenter. De vanlige knappene og menyene som finnes i nettleserne fjernes automatisk for å unngå forvirring.
Layouten benytter en bokmetafor. Oppstartssiden på Opsys er et bilde av en bok som er slått opp på første side. På høyre side i boken er alle kapitlene listet.

4.3 Tjenester i Opsys

Figur 4-2 Bilde av menyrammen. Knappen for søking etter ord er aktivert.

Alle knappene på menyrammen får et annet bilde når de blir aktivert. Bildet blir forandret til et liknende bilde som er "opplyst". Hvis knappen blir deaktivert, som for eksempel ved lukking av vinduet, forsvinner lyset igjen. I Figur 4-2 ser vi knappen for søketjenesten er aktivert.

4.3.1 Navigering



De to første knappene navigerer frem og tilbake på samme måte som vanlige nettleserne gjør. Siden nettleservinduet får den integrerte knappemenyen fjernet ved innlogging, er disse en erstatning. Grunnen til dette er at Opsys er designet til å fange opp lenkeklikking.

4.3.2 Lupen

Denne knappen åpner Lupen. Lupen vil bli presentert i Kapittel 5 og 6.

4.3.3 Søketjeneste

Denne knappen åpner et nytt vindu hvor en tjeneste for søking etter ord blir tilbudt. Dette vinduet består av tre deler: et skrivefelt hvor man kan skrive inn det ordet man vil søke etter, et indeksfelt som viser hvor i ordindeksen ordet befinner seg, og et resultatfelt som viser alle de dokumentene som har dette ordet.

Systemet søker rekursivt for hver bokstav. Dette betyr at når brukeren har tastet inn den første bokstaven, finner systemet det første ordet som begynner på akkurat den bokstaven. Hvis man for eksempel taster inn "f", vil systemet finne det første ordet som begynner på "f". Hvis man så taster inn "y" finner systemet det ordet som begynner på "fy", også videre. Hvis man taster inn "fysiskkkkk", det vil si at man taster "utenfor" rammen for et ord, vil systemet fremdeles vise det ordet som er mest korrekt. Indeksfeltet viser hele tiden det ordet som er nærmest i henhold til søkeordet. Indeksfeltet kan også navigeres direkte ved å benytte rullemenyen og klikke på selve ordene.

Figur 4-3 Søketjenesten

Resultatfeltet viser titlene til de dokumentene som inneholder termen det er søkt på. Resultatfeltet er blankt helt til et ord er klikket på i indeksfeltet, eller et ord er aktivisert i skrivefeltet. Hvis man klikker på et dokument i resultatfeltet, vil det dokumentet vises i presentasjonsrammen. Søkefunksjonen støtter de vanligste funksjonene med tabulator/klikking, og skifter automatisk over til engelsk hvis det er valgt i Konfigurasjonsknappen.

4.3.4 Merknader

For hvert dokument i Opsys, har brukerne mulighet til å lagre merknader. Merknader skal fungere som gule ”postit-lapper”. Merknader gjelder for det dokumentet som er gjeldende i presentasjonsvinduet. Når man trykker på merknadsknappen, får man opp et lite vindu som man kan skrive notater i. Når man er ferdig kan man lagre merknaden. Merknadsvinduet har også et ikon for å hoppe over til arbeidspaletten (Se 3.2.7). Hvis et dokument har en merknad, vises det ved et lite bilde av en penn og et papir i øverste høyre hjørne på dokumentet.

4.3.5 Video

Video er tilgjengelig via en ekstern, men integrert tjeneste. Når man trykker på videoknappen vises et lite vindu med en lenkemeny av alle videoene tilgjengelig. Hver video er en hel forelesning. Mens forelesningen spiller, vil aktuelle dokumenter fremvises i dokumentrammen. Dette er for å simulere en virkelig forelesing hvor foreleser snakker mens han/hun bruker en overhead eller skriver på tavlen. Ved klikking på en av linkene, forsvinner lenkemenyen, og blir erstattet av videoprogrammet Windows Media Player. Videoen begynner å spille automatisk. De vanligste funksjonene (stopp, start, pause og volumkontroll) til Windows Media Player er tilgjengelige som knapper under videobildet. Under videobildet er det også en rullegardinmeny. Denne rullgardinmenyen representerer stadier i den gjeldende forelesingen. Brukerne kan her hoppe frem og tilbake i forelesingen.

4.3.6 Arbeidspalett

I dette vinduet kan bruker se sine personlige merknader, markerte interessante dokumenter og meldinger fra faglærer.

4.3.7 Fle3

Fle3 er et samarbeidsverktøy som er utviklet av Universitet for Kunst og Design i Helsingfors. Knappen er en hyperlenke som åpner et nytt nettleservindu til Fle3. Vi skal ikke komme nærmere inn på Fle3 i denne oppgaven.

4.3.8 Dataleksikon

Denne knappen er en hyperlenke som åpner et nytt vindu til en leksikonside på Internett

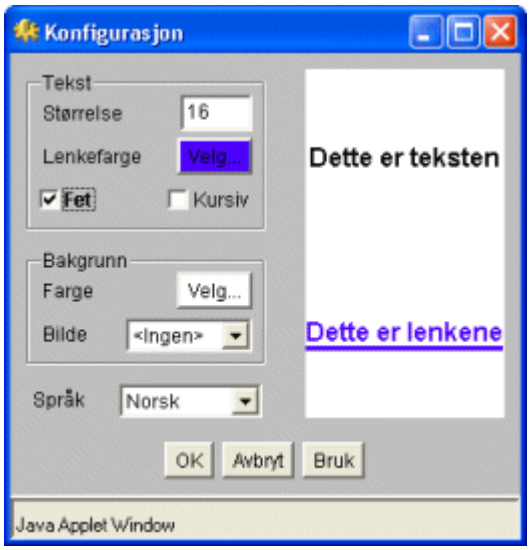
4.3.9 Stier

Opsys har støtte for navigering i stier. Man kan velge en sti, og så traversere den frem og tilbake i eget tempo. Man kan også direkte gå til en side ved å velge den i rullegardinmenyen. Det er to typer stier, personlige og forfatterbestemte. Personlige stier er stier som brukeren selv lager etter eget ønske. Forfatterbestemte stier er laget av faglærer, og kan ikke redigeres av brukerne.

Figur 4-4 Navigasjonsvindu for stier

4.3.10 Konfigurasjon

Brukerne kan selv bestemme utseende på dokumentene. Blant de parametere som de kan konfigurere er størrelse og utseende på teksten, farge på lenkene, bakgrunnsbilde og språk (norsk og engelsk).



Figur 4-5 Konfigurasjonsvindu for OPSYS

4.4 Arkitektur

Vi vil ikke gå i dybden på oppbyggingen av OPSYS. Vi skisserer kun de grunnleggende trekkene som får systemet til å fungere, og som er viktige å forstå for vårt eget program Lupen. For en mer detaljert beskrivelse av OPSYS kan man lese Warholms hovedoppgave[War00].

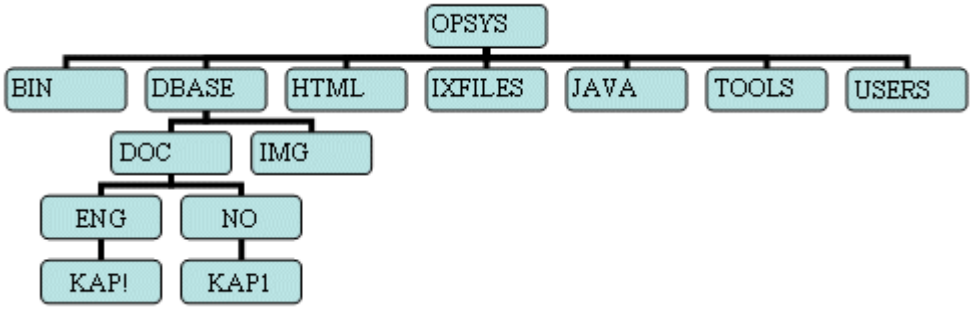
Opsys er implementert i JavaScript, Perl[28] og Java. Den opprinnelige versjonen lå på en UNIX-tjener. Apache[29] ble da brukt som webtjener. Under utviklingen av Lupen ble Opsys flyttet til en annen tjener (Mserver) som har Windows Server 2003 som operativsystem. På Mserver ble det integrerte Internet Information Services (IIS 5.0) brukt som webtjener.

4.4.1 Filstruktur

Katalogen *opsys* er hovedkatalogen for hele systemet. Under *dbase* ligger selve dokumentene, som er atskilt i norske og engelske dokumenter. Katalogen i*mg* inneholder alle bildene til Opsys. Katalogen *dbase* inneholder alle dokumenter og bildefiler som systemet trenger. De engelske og norske dokumentene er utskilt i

[28] Perl – Prectical Extraction and Report Language. Perl er et interpreterende skriptspråk, og følger CGI(Common Gateway Interface).
[29] Apache – En offentlig tilgjengelig webtjener. www.apache.org.

katalogene *eng* og *no*, og har en identisk filstruktur. Hvis brukeren vil skifte språk i Opsys, vil systemet da kun skifte ut *no* med *eng* i indeksene og ellers i programmene. Ingen andre hensyn til språk er de nødvendig internt i systemet. Bildefilene er de samme uansett språk, og disse trenger derfor ikke å være duplikater. Det eksisterer cirka 1000 dokumenter for hvert språk.



Figur 4-6 Overordnet filstruktur i OPSYS

- bin - Diverse kode i Perl.
- html - Systemets meny (og tips-) fil.
- users - Brukerkatalogene.
- java - Kode for alle Java appletene.
- tools - Diverse verktøy for oppdatering og vedlikehold av systemet.
- ixfiles - Indekser.

4.4.2 Indekser

Designet har gått ut i fra en filbasert løsning, i stedet for en databaseløsning. Arkitekturen har fulgt indeksrepresentasjonen fra tjeneren HyperWave[30]. I HyperWave er metadata utskilt fra selve dokumentet, og forutsetter at alle dokumenter har unike filnavn. En del av grunnen til dette er at dokumentet da kan flyttes til en ny posisjon, uten at lenkene i de berørte dokumenter blir ugyldige. Følgelig har alle dokumenter i Opsys unike filnavn, og metadata er organisert i ulike indekser.

30 [Mau96], http://www.iicm.edu/hgbook

Systemets hovedindeks: ”index.dat”

```
filnavn::dokument_nr::/kapittel/underkapittel/filnavn::norsk tittel:: engelsk tittel
```

- eksempel: Stakk_2.html::117::/kap1/maskinvare/Stakk_2.html::Stakk::Stack
- dokument_nr er en unik tallrepresentasjon av filnavnet.
- Perl-scriptet ”lagix.pl” i tools katalogen lager index.dat.

Søkeindeks: ”keyIXNO.dat” og ”keyIXENG.dat”

Disse to filene er stikkord- indekser for norske og engelske søkeord. Stikkordet har et tilhørende dokumentnummer (jf. hovedindeks). Søkeindeksen bygges opp ved traversering av dokumentstrukturen, hvor hvert dokument inneholder følgende HyperWave kompatible metadata-tagg:

```
<META NAME=”keyword” CONTENT=”stikkord1, stikkord2, stikkord3”>
```

Selve indeksen har da følgende oppbygging:

```
Norsk_stikkord::dokument_nr:dokument_nr….osv
```

- Eksempel: adgangsnøkkel::599
- Perl-scriptet ”lagMETA.pl” i tools katalogen lager disse to indeksfilene.

Lenkeindeks: ”lenkerIX.dat”

Dette er en indeks over alle lenker som finnes i nettverket.

```
Målfilnavn::lenkefilnavn1;lenkefilnavn2
```

```
Multiprogrammert.html::Minneorganisering_meny.html;Tresking.html
```

- Minneorganisering.html og Tresking.html har en referanse til Multiprogrammert.html
- Perl-scriptet ”lagIX.pl” i tools katalogen lager denne indeksfilen.

4.4.3 Overordnet virkemåte

Hovedprogrammet startes med scriptet *opsys.pl* i toppkatalogen. Dette scriptet er systemets hovedskript. Hovedscriptet kan ta inn flere argumenter i URL, og laster iht. argumentene forskjellige filer fra bin-katalogen. Hvis den f.eks. får argumentet *command=Body* vil den laste fila Body. I tillegg vil den kjøre en rutine med samme navn i sin respektive fil.

Når brukeren har logget seg inn med brukernavn og passord, kalles HTML filen *index.html*. Denne filen deler skjermen i to rammer, menyrammen og dokumentrammen. Hovedskriptet leverer så filen *Topp.html* til menyrammen og startdokumentet i Opsys til dokumentrammen. Menyrammen inneholder alle Java appletene og masse JavaScript kode.

Før hovedskriptet leverer et dokument til dokumentrammen vil det forandre dokumentets HTML-kode. Dokumentet har i utgangspunktet vanlige hyperlenker:

```
<A href="Hvorfor_stud_opsys.html">Hvorfor studere operativsystemer?</A>
```

Hovedskriptet vil da forandre denne vanlige HTML koden til:

```
<A HREF=javascript:parent.frames[0].ExecDoc('981')
onMouseOver="window.status='Innføring i operativsystemer - Hvorfor studere
operativsystemer?'; return true;" onMouseOut="window.status=''; return true;"
>Hvorfor studere operativsystemer?</A>
```

Det som skjer er at skriptet tar filnavnet (Hvorfor_stud_opsys.html), og leter opp det tilhørende unike dokumentnummeret i hovedindeksen (som her er 981). Den erstatter dokumentets vanlige lenke med et kall og et argument til JavaScript. Dette kallet går til menyrammen. I dette tilfellet kalles menyrammens JavaScript metode ”*ExecDoc”* med argumentet ”981”. ExecDoc tar i mot argumentet, og sender det videre til hovedscriptet. Hovedskriptet tar da og leverer det tilsvarende dokumentet tilbake til dokumentrammen.

Denne virkemåten gjør det mulig for appletene å vite hvilke lenker brukeren klikker på. I neste kapittel skal vi se hvordan vi kan gjøre det.

4.4.4 Sikkerhet

Som vi har sett må brukerne taste inn brukernavn og passord for å få tilgang til Opsys. Begge deler blir sendt til nett-tjeneren for verifisering. Brukernavnet (ikke passordet!) blir også fanget opp av miljøvariabelen ’*REMOTE_USER*’ i nettleseren. Hovedskriptet leser denne miljøvariabelen fra nettleseren, og benytter den videre. Blant annet legges brukernavnet inn i menyrammen sin variabel *USER*. Dette kan appletene benytte:

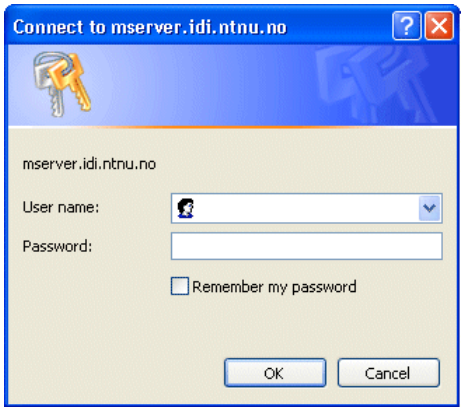
Før innlogging:

```
<PARAM NAME = "USER" VALUE = "--user--">
```

Etter innlogging:

```
<PARAM NAME = "USER" VALUE = "kåre">
```

Hver bruker får også en hjemmekatalog under user-katalogen, og den får samme navn som brukeren. Opprettelse av kataloger blir gjort av webmaster. De forskjellige katalogene i systemet får tildelt en sikkerhetsmodus av webmaster som gir lesetilgang og/eller skrivetilgang. I utgangspunktet har alle brukerkatalogene kun lesetilgang.



Figur 4-7 Innloggingsvindu for Opsys

4.4.5 Appleter

Før utviklingen av Lupen var det to appleter i Opsys: Navigasjonsappleten som gav sti-funksjonaliteten, og Konfigurasjonsappleten for endring av utseendet på dokumentene. Begge var utviklet i Java versjon 1.1, og ligger i en Jar-pakke ”*mnfit222.jar*” som ligger i Java-katalogen.

Sikkerhetsmekanismene til Java gir ikke tilgang for appleter å skrive til disk med mindre appleten er signert med en digital signatur. Ingen av appletene er signert, da dette må gjøres av en tredjepart som for eksempel firmaet Verisign, og koster masse penger. Appletene kan heller ikke lese miljøvariablene til nettleseren, og vet derfor i utgangspunktet heller ikke hvem brukeren er. Dette er naturligvis et problem, da begge appletene må lagre stier og personlig konfigurasjon på brukernes hjemmekatalog.

Dette problemet løses på en fiffig måte. Siden Appletene har tilgang til å lese sin egen HTML-fil (menyrammen), kan de lese variabelen ”USER”, og vet derfor hvem brukeren er. Perl-skript har full tilgang til å skrive til tjeneren. Appletene samler så all tekst de vil ha skrevet til tjeneren, pluss eventuelle kommandoer, inn i en enkel linje med tekst. Deretter åpner de en forbindelse til et Perl-skript, og gir dette tekstlinjen og brukernavnet som kommando. Skriptet skriver så til filen. Vi skal de på dette i mer detalj i neste kapittel.

5 Prototype på et dynamisk nettkart

5.1 Innledning

I dette kapittelet vil vi vise hvordan vi la grunnlaget for Lupen. Som vi har nevnt før i oppgaven, baserte vi oss på forandringer og utvikling etter hvert som vi skred fram. Dette kapittelet er ment som en klargjøring av alt arbeidet som lå bak den endelige versjonen av Lupen.

Prototypen kom som svar på et ønske om å tilføre det eksisterende systemet (Opsys) et dynamisk nettkart. Dette skulle være et tillegg til de tekstbaserte navigasjonsprogrammene som allerede var implementert. Fokus skulle ligge på brukervennlighet, plattformuavhengighet og støtte for multimedia. Oppgaven hadde ellers ingen spesifikasjoner. Spesifikasjonene måtte vi utvikle mens arbeidet gikk framover. Det var uvisst hva som var mulig å gjennomføre rent teknisk sett. Vi var derfor åpne for at prosjektet kunne forandre seg radikalt under arbeidet.

5.2 Identifisering av spesifikasjoner

Systemet Opsys er basert på faget mnfit222 Operativsystemer, og er et mellomfag. Mange brukere som tar slike fag er relativt erfarne med datamaskiner. Det må forutsettes at mange av dem benytter en annen plattform enn Windows, f. eks Unix, Linux eller Macintosh. Et datafag på NTNU bør ta hensyn til dette, og ikke eksplisitt ekskludere brukere av andre systemer enn Windows. Opsys støtter både engelsk og norsk språk, og det er ønskelig å videreføre dette også i Lupen.

Siden Opsys er laget med formål å støtte fjernundervisning er det ikke ønskelig at brukerne skal belemres med nedlastninger. Responstiden bør også holdes på et minimum. Mange studenter har tilgang til raske nettoppkoblinger som ADSL, ISDN og LAN, men pr. dags dato har ikke alle brukere tilgang til dette. Det forutsettes at noen av brukerne bruker gamle modemer med lav hastighet, og dette skal det taes hensyn til.

Som nevnt i tidligere kapittel inneholder hypermedia også bilder, video og lyd. Det er ønskelig at prototypen også skal ta hensyn til dette.

Faget mnfit222 Operativsystemer er et mellomfag på Institutt for Datateknikk og Informasjonsvitenskap, og derfor har brukerne per definisjon et relativt høyt kompetansenivå innen IT. Uansett er det ønskelig at brukergrensesnitter er enkelt og oversiktlig. Internets popularitet ligger i dets enkelhet og brukervennlighet, og dette bør taes hensyn til.

I dag har Internett Explorer over 90 % av nettlesermarkedet. De 10 prosentene som er igjen utgjør fremdeles et enormt antall brukere pga. den eksplosive veksten av Internett. Disse nettleserne har ganske ulike standarder og man er nødt å velge en standard som man kan gå ut ifra. Det mest hensiktsmessige var å gå ut ifra den mest brukte. Internet Explorer 5 ble valgt som mal, og tidligere versjoner ble ignorert.

De seks spesifikasjonene som er identifisert er:

- Programmet skal være plattformsuavhengig
- Støtte for både norsk og engelsk
- Rask nedlasting.
- Støtte for hypermedia.
- Brukervennlighet
- Støtter Internet Explorer 5 og nyere.

5.3 De første undersøkelsene

5.3.1 Java som programmeringsspråk

Java ble raskt valgt som programmeringsspråk fordi det allerede hadde blitt brukt i de to andre appletene. Med Java ville spesifikasjonen om plattformuavhengighet også bli oppfylt. I tillegg hadde det en grei læringskurve og relativt god støtte for grafikk. De andre forslagene som ble undersøkt var Delphi og C++. Ingen av disse språkene er laget spesifikt med tanke på Internett. Disse ville også brutt med plattformuavhengigheten. De ble raskt forkastet til fordel for Java.

De to andre appletene (Konfigurasjonsappleten og Navigasjonsappleten) var utviklet med Java versjon 1.1. Dette var en gammel standard som benyttet AWT[31] for de grafiske brukergrensesnittkomponentene. AWT benyttet Java Virtual Machine (JVM) som var installert på de fleste plattformer. Hver plattform brukte forskjellige typer JVM, noe som gjorde at samme Java program så litt ulik på de forskjellige plattformene. Windows 98, ME, og XP (før Service Pakke 1) ble levert med JVM installert. Disse trengte derfor ikke å laste ned noen ekstra programvare for å kjøre Java. Ulempen med å bruke AWT er at utviklerne må ha mer detaljert kontroll over detaljer og komponenter enn senere versjoner av Java. Komponentene er store og ”tunge” til forskjell fra nyere versjoner som er lettere og enklere. En slik ny versjon er Swing.

Swing benytter sitt eget bibliotek, noe som gjør at samme program ser likt ut de forskjellige plattformene. Ulempen er at man må laste ned egen programvare for å kjøre Java programmer som benytter Swing. Komponenter fra Swing er enklere og lettere å utvikle enn AWT. Appleter som benytter AWT er fremdeles kjørbare i Swing.

Vi ville ikke at brukerne måtte laste ned Swing for å kjøre vår applet, siden en av spesifikasjonene var rask nedlasting. Men siden Swing var enklere og lettere å starte med, valgte vi likevel det. Holdningen var at hvis ikke Swing fungerte, ville det være en relativt enkel ting å modifisere appleten tilbake til AWT.

Utviklingsverktøyet som ble benyttet var JBuilder Enterprise 7.

5.3.2 Sirkelbasert layout

Selve dokumentbasen var arrangert hierarkisk. Fagpensumet er organisert i kapitler og underkapitler, og denne strukturen var videreført i dokumentbasen. Men basen har også lenker som går på kryss og tvers som i en vanlig nettverksstruktur. Som vi så i WebTOC ville disse lenkene lage et uhensiktsmessig rot hvis vi skulle brukt en tremodell.

[31] Abstract Window Toolkit. AWT er i ferd med å bli faset ut i Java.

Vi ble imponert av Star Three sin hyperboolske layout. Man kunne vise et stort avtall noder på et relativt lite område, men samtidlig ikke miste følelsen av hvor man er. Dette kunne passe godt inn i Opsys, som har cirka 1000 dokumenter og 7-8 nivåer med i strukturen. Potensielt kunne vi vise hele systemet på et eneste skjermbilde. Problemet med denne layouten er at matematikken er uhyre kompleks, kanskje utenfor rekkevidde for oss. Den var i alle fall ikke noe å starte med.

Den sirkelbaserte layouten som Mappucino bruker, var ikke like god som Inxight. Nodene blir ikke mindre mot horisonten, og dermed får vi ikke plass til å vise alle nodene på samme tid. Dette gjør at man må avveie programmets størrelse på skjermen, mot antall nivåer som blir vist. Likevel ville de være enklere å vise lenkene mellom noder på forskjellige nivåer enn slik som på WebTOC. En sirkelbasert layout ville også støtte sirkulære lenker. Layouten virket også enklere matematisk, og innenfor vår rekkevidde.

Vi gikk til slutt for Mappucinos layout som en inngangsbillett for å starte utviklingen av en prototype. Vi hadde i tankene at vi kunne forandre layouten underveis, hvis det ikke fungerte som ønsket. I utgangspunktet gikk vi for en visning av 3 nivåer, plassert som konsentriske sirkler.

5.3.3 Spesifisering av edderkopp

Som vi har sett kan en edderkopp traversere ett nettsted og samle inn informasjon. For å legge ut nodene og lenkene på det grafiske nettkartet, trengte også vi å samle inn informasjon om hypermediebasen til Opsys. Vi trengte å vite hvilke lenker som eksisterte mellom de forskjellige dokumentene, og hvordan de er plassert i forhold til hverandre. Samtidlig måtte vi samle inn informasjon om de forskjellige multimediefilene som eksisterte i hypermediebasen. På Opsys er delen som omhandler video utskilt på en egen server. Video er ikke implementert i selve dokumentene, så vi valgte bort den delen. Derfor trengte ikke edderkoppen samle inn annen multimedieinformasjon fra dokumentene enn bilder og animasjoner.

Vi hadde også et ønske om å trekke ut de lenkene som førte ut i fra selve Opsys og til det generelle Internett. Vi ville eksplisitt markere disse lenkene for brukeren.

I stedet for å utvikle en egen edderkopp som traverserte hypermediebasen ut i fra en start-URL, kunne vi bruke de forskjellige indeksfilene som allerede eksisterte. Dette ville spare oss for å utvikle en egen edderkopp basert på URL. Vi trengte bare å finne en metode for å ”trekke ut” strukturen fra indeksfilene.

Innsamlingen av bilder, animasjoner og eksterne lenker kunne bli gjort ved å søke direkte i dokumentene, og bygge opp nye indeksfiler. Hvis disse filene fulgte samme standard som de andre indeksfilene, kunne vi bygge opp et nettkart som hentet ut all informasjon om hypermediebasen fra katalogen til indeksfilene..

På grunnlag av dette så vi at vi trengte to ting:

- En motor som genererte en representasjon av nettverket ut i fra de eksisterende indeksfilene.
- Et program som søkte etter bilder, animasjoner og eksterne hyperlenker i HTML koden til alle dokumentene.

5.3.4 Ikke-tilstandsløst nettkart

Bruken av JavaScript og dynamisk generering av HTML koden i dokumentene gav oss en mulighet for å knytte Lupen til brukerens klikk i nettleseren. Dette var noe vi ville utnytte. Ideen var at hvis brukeren klikker på en lenke i nettleseren, kan Lupen reagere på dette. Dette kunne brukes til å bygge opp en konsis historie over brukerens bruk av Opsys: Hvilke sider han har besøkt og ikke besøkt. Senere ville vi også bruke denne tjenesten til å bygge opp en historie som ble lagret på brukerens hjemmeområde. Dermed kunne Lupen huske historien utover en vanlig sesjon av nettleseren.

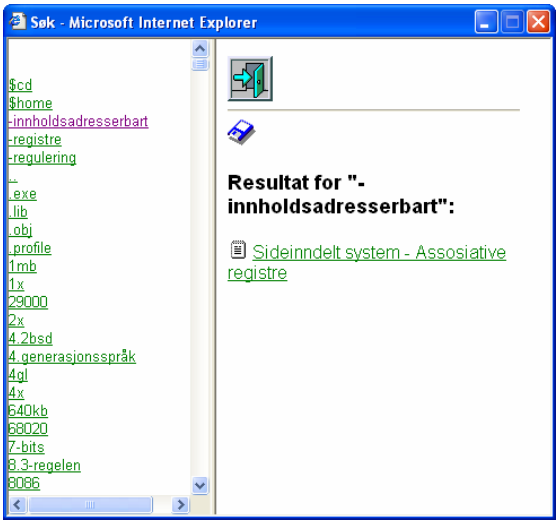
5.3.5 Andre aspekter

Visningen av nodene skulle begynne som ikoner. Disse ikonene skulle da forandre farge når som de ble klikket på. Dette ble sett på som relativt enkelt, og forhindret ikke en mer avansert visning senere. Det viktigste var i begynnelsen å få utviklet en prototype som vi kunne arbeide ut i fra.

Vi ville også at nodene på en eller annen måte skulle vise om dokumentene inneholdt bilder, animasjoner og eksterne hyperlenker. Tanken var at nodene kunne ha forskjellige små ikoner for å markere dette.

5.4 Testkjøring: Søkeapplet

Helt fra starten av prosjektet hadde vi et ønske om å utvide søkefunksjonaliteten i Opsys. Den var opprinnelig basert på Perl og dynamisk generering av HTML-dokumenter.



Figur 5-1 Det opprinnelige søkevinduet.

Vi ville utvide funksjonaliteten med en skrivefunksjon (se 4.3.3). Dette ville vi gjøre i Java. Siden klikking på ord ville få Opsys til å laste inn det aktuelle dokumentet, ville dette bli en test på integrering av appleter i Opsys. Vi hadde liten erfaring med Java og teknikken bak Opsys, og dette ville gi oss en innføring i begge deler.

Selve grensesnitter var enkelt å utvikle. Vi brukte AWT komponenter, og leste inn indeksfilene *keyIXNO.dat*, *keyIXENG.dat* og *index.dat*. Disse filene ble lest fra serveren igjennom bruk av HTTP-protokollen.

5.4.1 Liveconnect

For å avgjøre hvilket språk brukeren hadde valgt (norsk var standard), måtte vi få tilgang til JavaScript metodene på menyrammen. Java appleter som ble kjørt i Netscape kunne gjøre dette som standard fordi Netscape benyttet sin egen pakke **LiveConnect**. Ved bruk av Java i Internet Explorer må man eksplisitt importere LiveConnect inn i Java. Selve LiveConnect-filene må da ligge i jar-pakken til appletene.

For å bestemme hvilket språk(engelsk eller norsk) som benyttes i Opsys, må Søkeappleten spørre menyrammen om dette: Hver gang appleten starter opp, blir en metode *callJS* kjørt. CallJS tar inn et metodenavn og eventuelle parametere, og bruker Liveconnect til å kalle opp den aktuelle metoden i menyrammen. Menyrammen kjører metoden, og returnerer resultatet (Hvis metoden har noe resultat). I dette tilfellet kaller vi opp metoden *getLanguage*:

```
//importerer LiveConnect
import netscape.javascript.*;

//denne metoden kaller opp callJS metoden som er felles for all
//kommunikasjon mellom Java og Javascript.
protected void launchFrame() {
    String lang = callJS("getLanguage", null);
     ....
}

//tar som argument metodenavnet og parameteret som sendes til JavaScript.
public String callJS(String func, String[] args) {
      JSObject win = JSObject.getWindow(this);
      Object o = win.call(func, args);
      if (o != null) {
        return o.toString();
      }
      return null;
    } catch(Exception e) { e.printtackTrace();
      return null; }
  }

//JavaScript metode i menyrammen som returnerer en variabel 'no' eller 'en'
//avhengig av valgt språk.
function getLanguage() {
  if (language == 'no') {
    return 'no';
  }
  else if (language == 'eng') {
    return 'en';
  }
}
```

Variabelen som returneres viser språket som blir benyttet. Denne brukte vi til å lese enten den engelske indeksfilen (keyIXENG.dat) eller den norske (keyIXNO.dat).

5.4.2 Bruk av indekser

Hvis for eksempel brukeren taster inn ordet ”adresserommet” ser vi i fra filen keyIXNO.dat at ordet finnes i dokumentnummer ”909”.

```
adresserommet::909
```

Tittelen på dokument 909 (adresserommet) blir så vist i resultatfeltet. Hvis brukerne klikker på denne tittelen igjen, benytter vi igjen callJS for å kalle metoden ExecDoc i menyrammen med dokumentnummeret som argument:

```
//metode som kaller opp Javascripts metode ExecDoc med dok_nr som argument
public void showPage(int n){
  String[] Args = new String[1];
  Integer x = new Integer(n);
  Args[0] = x.toString();
  callJS("ExecDoc",Args);
}
```

ExecDoc viser da dokumentet i dokumentrammen.

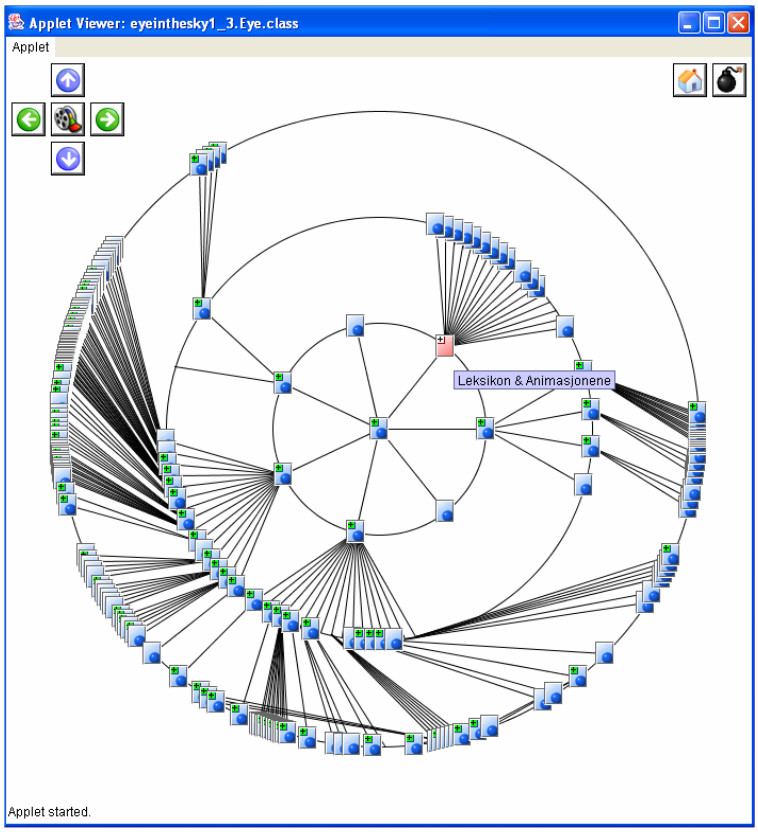
5.5 Første prototype

5.5.1 Bakgrunn til første prototype

Vi ville lage en prototype umiddelbart. Derfor laget vi et eget program **FileParser** for å ekstrahere nettverket i fra indeksfilene. Resultatet var en fil som innehold alle lenkene til alle dokumentene i følgende format:

```
1::2
2::56:66
4:23:7:55
```

Dok1 lenker til dok2. Dok2 lenker til dok56 og dok66 og så videre. Vi hadde også utviklet en matematisk algoritme for å plassere nodene ut på skjermen (vi skal se nærmere på den senere). Prototypen var et statisk skjermbilde, uten noen funksjonalitet. Nodene var implementert som JButtons, en standard komponent i Swing. For å tegne lenken i mellom nodene, trakk vi ut XY-koordinatene til de nodene som lenken gikk i mellom. Disse koordinatene ble lagret i en matrise. En egen metode for tegning av lenkene ble implementert i en hoved-klasse, ikke i klassen til nodene, Page.



Figur 5-2 Første prototype av Lupen

Etter at prototypen var ferdig satt vi igjen med følgende relevante klasser:

- Eye – startklassen. Denne initierte Manager
- Manager – Beregner XY-koordinatene, tegner streker og legger ut noder på GUI.
- GUI – Panel (tavlen) som nodene og strekene tegnes på. Styrer tegning.
- Page – representasjon av de forskjellige nodene

5.5.2 Grafiske utfordringer

Java er et objektorientert programmeringsspråk. I utgangspunktet skal alle objekter selv være ansvarlige for å tegne seg selv, og å bevare sin egen tilstand.

Forholdet mellom nodene og lenkene skapte en del hodebry. Prototypen baserte seg på at Manager beregnet og gav XY-koordinatene til nodene, og la nodene ut på GUI. Deretter tegnet nodene seg selv på GUI. Men lenkene i mellom dem ble tegnet av Manager. Dette gav det resultatet at tegningen av lenkene ikke ble oppdatert korrekt.

Lenker som ble overlappet av andre vinduer ble ikke oppdatert når disse vinduene ble fjernet igjen. Vi prøvde å tvinge inn oppdatering, men det ble klart at denne løsningen ikke holdt vann. Nodene måtte selv være ansvarlige for sine egne streker. Men dette ville igjen føre til at en lenke som gikk imellom node A og B, ville bli tegnet av begge. Vi måtte finne en måte å dele kontrollen av strekene mellom de forskjellige nodene.

Vi innså at fordelingen av lenker i mellom dokumentene i hypermediebasen var svært ujevnt fordelt. Nodene klumpet seg sammen på visse områder. For å overkomme dette var det ønskelig å forstørre og forminske appleten. Men den matematiske ligningen gikk ut i fra en perfekt sirkel, ikke en oval. Hvis vinduet ble forstørret eller forminsket på en rektangulær måte ville det gi problemer. Nodene kunne da forsvinne utenfor skjermbildet. Vi kunne utvikle ligningen til å bli tilpasset en oval, som ville gjøre det mulig å forstørre vinduet rektangulært. Vi undersøkte mulighetene for dette ved Matematisk Institutt på NTNU, men ble forklart at dette var en for stor oppgave.
Vi måtte derfor tvinge en eventuell forstørrelse til å være en perfekt firkant.

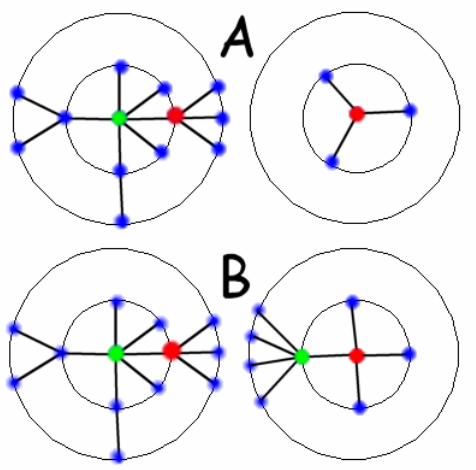
Nodene var en utvidelse (extends) av Swings JButton-klasse. Bruken av slike komponenter var tregt, og kompliserte forholdet til tegningen. Det ble klart at for å ha full kontroll over grafikken, måtte vi gå vekk fra en komponentbasert grafikk. I stedet måtte vi lage våre egne klasser som gav oss full kontroll og forståelse for grafikken. Swing ble derfor forkastet til fordel for AWT. AWT er mer tungvint, men tvinger utvikleren å forstå og kontrollere alt på detaljnivå. Vi ville selv tegne alt som skjedde på panelet. En egenkomponert tegning av alle de visuelle delene i appleten ville gi oss frihet for å utvikle og forandre grafikken. Dette ville gi en inngående forståelse av Javas grafiske arkitektur, og også potensielt nye muligheter.

5.5.3 Kognitive utfordringer

Vi gikk ut i fra en størrelse på appleten på mellom 400 og 700 piksler. Grunntanken var at hvis appleten var større ville den virke ”påtrengende” på skjermen. Det ville ikke gi følelsen av et hjelpsomt og enkelt nettkart. Hvis den ble mindre enn 400 piksler ville den være for liten for å vise nodene på en meningsfull måte. Vi valgte derfor 700 som standardstørrelse, med mulighet for forminskning til 400, og forstørring til 1000.

Vi trodde at en visning utover 3 nivåer eller sirkler ikke ville være hensiktsmessig. I så fall måtte nodene bli mindre, eller vinduet større. Ved 3 nivåer kunne vi også gi brukeren et valg om å forstørre nodene en del, uten å forstyrre skjermbildet nevneverdig. Da kunne nodene også vise deler at tittelen på dokumentene. Uten titlene ville det være vanskelig for brukeren se forskjellen mellom dokumentene. Hvis brukeren likevel ville se små noder, kunne vi vise tittelen som et verktøytips når han bevegde musen over dem.

Vi måtte bestemme oss for hvordan brukeren navigerer seg fram og tilbake blant nodene når han klikker på dem. Vi bestemte oss for at den gjeldende noden hele tiden må være i origo. Hvis det ikke var det, ville det virke inkonsekvent. Den viktigste noden (den gjeldende) burde naturligvis ligge på den mest åpenbare plassen (origo). Derfor måtte de omkringliggende nodene vise de dokumentene som noden lenket til. Dette gav oss et nytt problem, da skjermbildet ville forandre seg dramatisk mellom klikkene. Brukeren ville ikke se noen kognitiv sammenheng mellom den ene tilstanden og den andre. Orienteringen ville bli forstyrret av flere svært ulike skjermbilder etter hverandre.



Figur 5-3 A: Mister forrige node. B: Bevarer forrige node.

I Figur 5-3 ser vi i eksempel A hvordan dette fungerer. Brukeren befinner seg på GRØNN, og klikker så på RØD. Sammenhengen til GRØNN går tapt, og skjermbildet er totalt forskjellig. Brukeren kan ikke se noen sammenheng i nettverket. Figur B viser hvordan man potensielt kan ta vare på den forrige noden. Når brukeren klikker RØD, vil det andre skjermbildet fremdeles også vise GRØNN, og brukeren vil se sammenhengen

mellom disse to tilstandene. I dette tilfellet er også grønn plassert på venstre side, noe som indikerer ”tilbake”.

Logikken bak tilfelle B er relativt enkel; man reserverer en del av venstre side for det dokumentet (GRØNN) som lenker til RØD. Man bruker også en del av de påfølgende sirklene til å vise GRØNN sine noder. Disse kan være både de nodene som lenker *til* GRØNN, og de som lenker *fra* GRØNN. Vi ville bruke de muligheter som tilfelle B ville gitt. Men vi valgte å basere oss på tilfelle A til å begynne med. Dette fordi vi trodde vi senere i utviklingen hvor vi hadde mer kunnskap, kunne gå tilbake til tilfelle B.

Strekene som indikerte lenkene var noe mangelfull. Blant annet ville vi indikere hvilke noder som lenket TIL og hvilke som lenket FRA. Vi besluttet å vise retningen med en pil. Brukeren ville også se en nodes lenker bedre hvis disse forandret farge når musen ble beveget over dem.

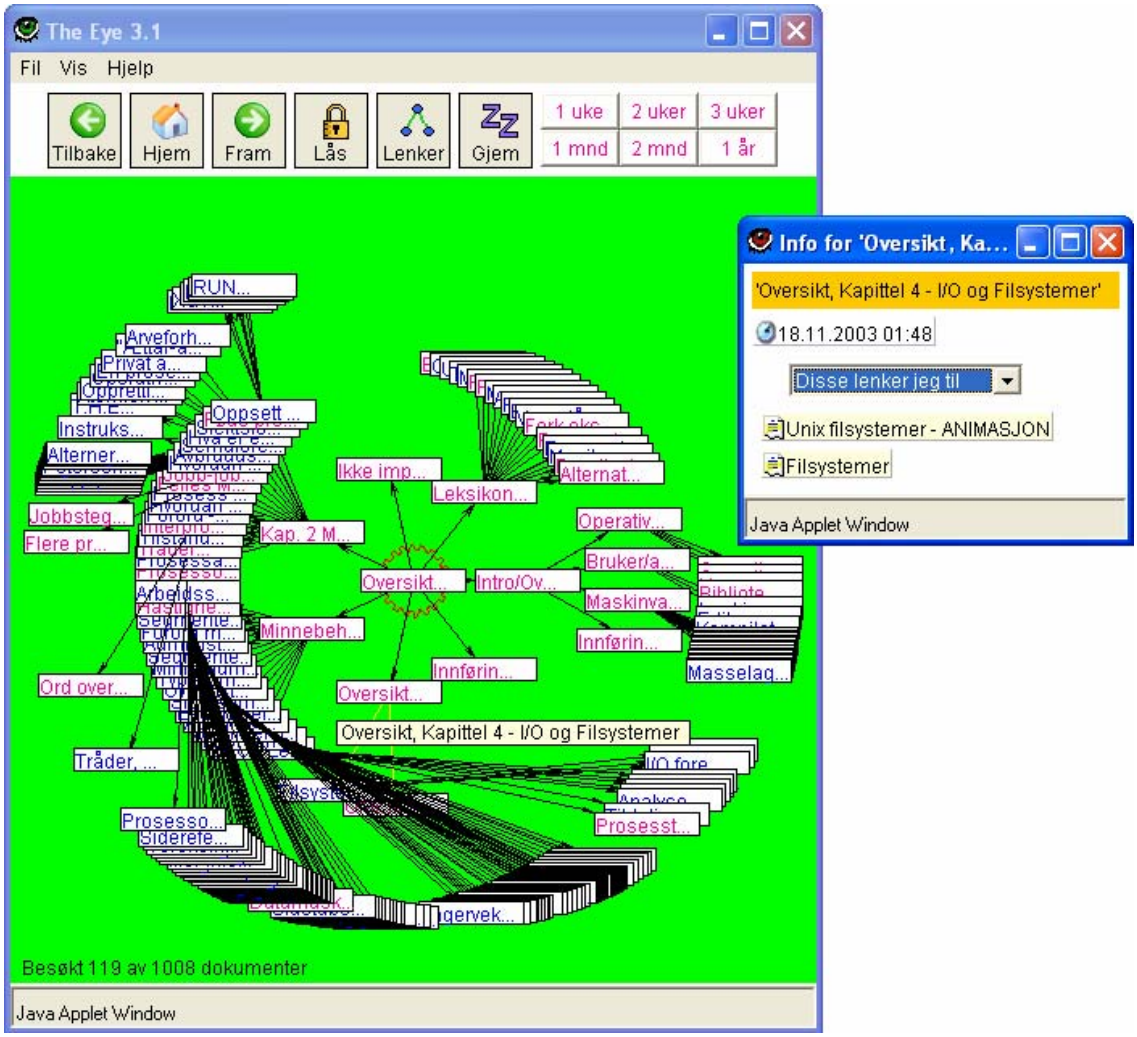
Siden nodene var relativt små og tittelen ville ta opp det meste av den tilgjengelige plassen, ville vi gi brukeren mer detaljer: Hvis brukeren høyreklikket på et ikon, kunne han åpne et vindu som viste full tittel, eksterne og interne lenker (i Opsys), bilder og animasjoner.

Antall dokumenter i hypermediebasen var også viktig å vise, Brukeren kunne dermed få en ide over størrelsen, siden nettkartet ikke ville vise alle noder på en gang. Dette kunne vises ved hjelp av tekst på selve GUI.

Siden nodene kunne bli samlet i klynger, kunne det bli vanskelig å skille ut enkeltnoder. Dette problemet kunne omgåes ved at brukeren kunne flytte om på nodene i skjermbildet. Hvis brukeren flyttet mye omkring på nodene kunne dette føre til at brukeren mistet forståelsen av hvor noden opprinnelig befant seg. Dette kunne vi motvirke ved at brukeren kunne klikke på noden, og noden ville selv gå tilbake til den opprinnelige posisjonen.

Det kunne potensielt bli for mange lenker på kryss og tvers. Dette ville forvanske skjermbildet. Vi ønsket derfor at brukerne skulle kunne velge mellom å vise alle lenker, eller bare vise de ikke-sykliske.

6 Ferdig versjon av Lupen



Figur 6-1 Oppstartsbilde av Lupen

6.1 Innledning

I dette kapittelet vil vi vise hvordan Lupen er bygd opp, og hvordan programmet fungerer. Vi har delt kapittelet opp i deler som belyser de viktigste aspekter av Lupen på en logisk måte. Utviklingen skjedde naturligvis ikke på denne oppdelte måten, men presentasjonsmåten er gjort med hensyn til leseren.

Lupen er kompatibel med alle versjoner av Java etter versjon 1.3.1.

6.2 Funksjonalitet

6.2.1 Generell funksjonalitet

Oppstart av Lupen skjer ved at brukeren trykker på en knapp i Menyrammen. I Figur 6-1 ser vi vinduet som blir generert. Brukeren kan nå trykke på en node, og det gjeldende dokumentet vil bli lastet i nettleserens dokumentramme. Hvis brukeren klikker på en lenke i selve nettleseren, vil Lupen hoppe til det dokumentet igjen. Hvis musen beveger seg over en node, vil den fulle tittelen til dokumentet vises i et eget, gult felt like ved noden. Det lille vinduet til høyre på figuren, indikerer at brukeren har høyreklikket på en node. Dette åpner et nytt vindu, hvor mer detaljert informasjon om dokumentet blir vist. Aktuelle ikoner og tilhørende informasjon er:

-  Et bilde eksisterer. Viser også filnavn.
-  Tidspunkt for sist besøk.
-  Eksterne lenker ut i fra Opsys. Viser også URL-adressen
-  Animasjoner.
-  Dokumentets lenker til andre dokumenter på Opsys, viser dokumenttittel.
-  Dokumenter som lenker til det gjeldende dokument.

6.2.2 Historie

Hvis brukeren allerede har besøkt en side, vil noden forandre farge. Man kan selv velge hvilken farge. I tillegg kan man sette farge i henhold til tidsperioder. I Figur 6-2 ser vi en del av dette skjermbildet. Man kan for eksempel velge rød farge på alle noder som er blitt besøkt siste uke.

Figur 6-2 Valg av farge på besøkte noder

Knappene for fram, tilbake og hjem fungerer på samme måte som en vanlig nettleser. De er også synkronisert med de samme knappene i Opsys. Dette vil si at hvis brukeren trykker på tilbakeknappen i Opsys, vil dette kopieres i Lupen.
Informasjon om besøkte sider lagres på brukerens katalog i Opsys. I nederste, venstre hjørne på Figur 6-1 ser vi hvor mange dokumenter brukeren har besøkt.

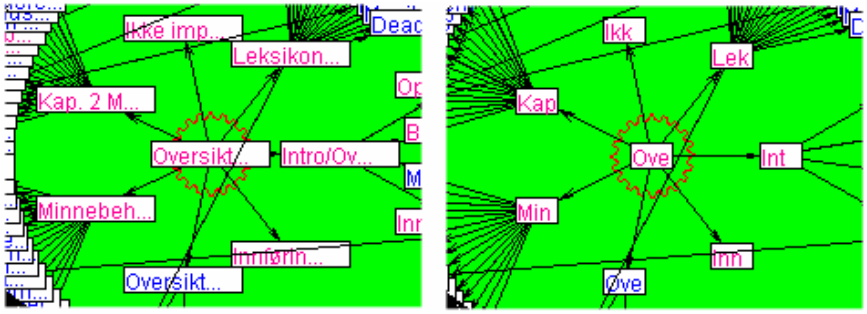
6.2.3 Lås

Siden Opsys er pensum, vil gjerne brukerne se hvor mye av innholdet i som er lest. En node som er markert som besøkt (forandret farge), vil gjerne indikere at siden faktisk er lest. Vi har lagd låsfunksjonen for å gi brukeren et valg om nodene skal bli markert som besøkt eller ikke. Dette vil gjerne være aktuelt hvis brukeren bare skal surfe seg rundt på Lupen for å få en oversikt over innholdet. Den tiden Låsen er slått av, vil nodene han klikker på ikke bli markert som besøkt. Ved klikk på nodene, vil heller ikke siden vises i Opsys. Dette vil oppmuntre utforskning av innholdet, uten å miste forståelsen av hvilke sider som faktisk er blitt lest. Når brukeren slår på låsen igjen, vil Lupen hoppe til den siste noden som var besøkt før låsen ble slått av.

Hvis brukeren klikker på lenkene i Opsys direkte mens låsen er av, vil disse likevel markeres som besøkte i Lupen. Dette er fordi brukerne er vant til at klikk på lenker i nettleser vil markere disse som besøkte. Det er standard oppførsel for en nettleser. Hvis dette ikke skjedde ville nettleseren markere siden som besøkt, men ikke Lupen. Dette ville da ikke være konsist, og gi opphav til forvirring.

6.2.4 To typer noder

I menyen til Lupen kan man velge om man vil se nodene som små ikoner, eller større rektangler. Begge visningene viser litt av tittelen til dokumentet. De to størrelsene er hardkodet i Lupen, og vil ikke bli større hvis Lupen blir maksimert eller minimert



Figur 6-3 Visning av nodene som rektangler eller ikoner

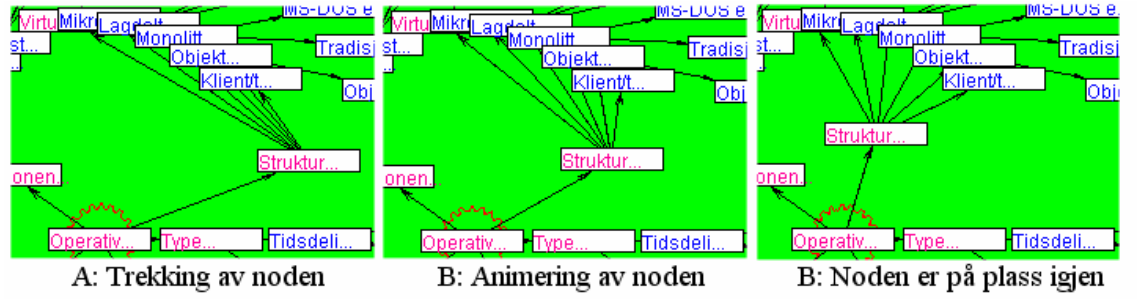
6.2.5 Lenkevisning

Brukeren kan ved å klikke på knappen for lenkevisning, velge om han vil se bare ikke-sykliske lenker, eller alle lenker. Ved et klikk vil Lupen oppdatere seg øyeblikkelig, og gjenspeile brukerens valg. Standard er kun visning av ikke-sykliske lenker.

Når brukeren fører musen over en node vil alle lenkene til noden forandre farge til gul. De blir ”opplyst”. Funksjonaliteten er samordnet med lenkevisningen, slik at den opplyser kun de lenkene som brukeren har valgt (se avsnittet ovenfor).

6.2.6 Animasjon

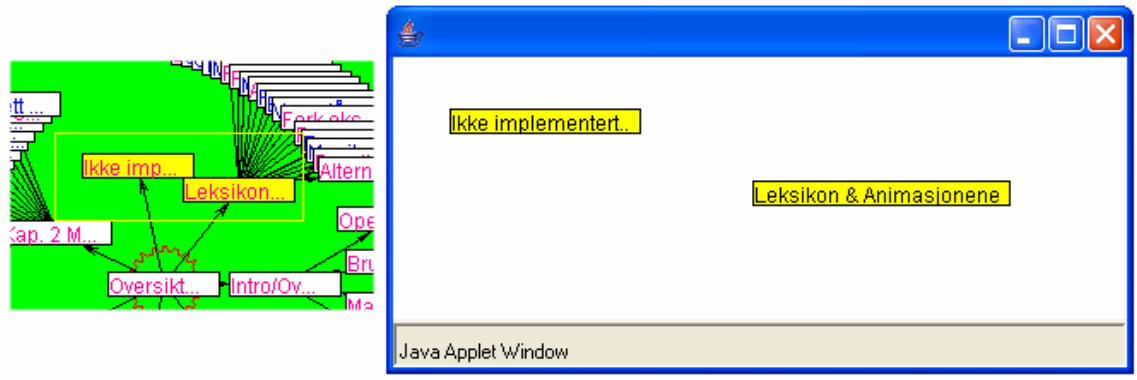
Brukeren kan venstreklikke og trekke i alle noder. Noden kan ikke trekkes utenfor vinduet. Hvis brukeren klikker på noden igjen, vil den gå tilbake til den opprinnelige posisjonen. Man kan posisjonere flere noder på samme gang. Det kan også være flere noder som samtidig animerer seg tilbake til sin opprinnelige posisjon. Hvis animasjon pågår, vil dette vises ved at et lite hjul rundt origo vil snurre rundt.



Figur 6-4 Eksempel på trekking og animering av en node

6.2.7 Forstørrelse av område

Brukeren kan forstørre et område på Lupen. Dette er fordi nodene kan potensielt ligge meget tett opp til hverandre, og overlappe hverandre. Brukeren kan da venstreklikke på et tomt område, og dermed trekke ut et rektangel. Alle noder som blir omsluttet av rektangelet, vil forandre farge til gul. Dette indikerer at de er blitt valgt. Når brukeren ”slipper opp” venstreklikket, vil et nytt vindu skapes. Dette vinduet er en forstørrelse av det valgte området.



Figur 6-5 Eksempel på forstørrelse av et område.

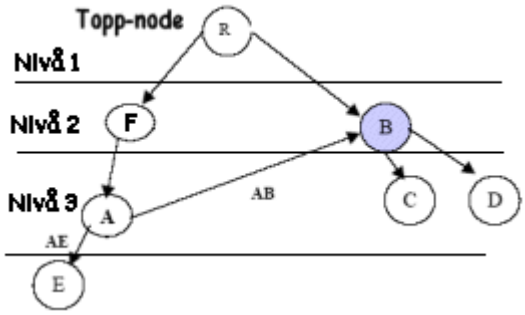
Funksjonaliteten er ikke blitt ferdig utviklet. Blant annet mangler nodene i det forstørrede vinduet lenker, og de er heller ikke klikkbare.

6.3 Oppstart: Indeksering

6.3.1 Nettverksstruktur

Ved oppstart av Opsys, starter Lupen å bygge opp en indeks som representerer nettverksstrukturen. Siden Opsys’ hypermediebase har en hierarkisk struktur, kan alle dokumenter nåes ut i fra en toppnode. Filnavnet til toppnoden blir gitt Lupen som et parameter. I vårt tilfelle er filnavnet ”*intro.html*”. I tillegg leser Lupen inn indeksfilene index.dat og lenkerIXNO.dat. Dette er alt som trenges for å bygge opp indeksen.

Klassen som er ansvarlig for bygging av indeksen heter **PageMaker**. Den oppretter en tom matrise som inneholder alle nodene. Alle noder er et tilfelle av Klassen **Page**.



Figur 6-6 Oppbygging av Indeks

For å representere nettverket, må vi sette et kunstig skille mellom lenkene. Som vi ser i Figur 6-6 lenker R til B og F. Vi sier at R er en **far**, og F og B er **barn**. Sykliske lenker kompliserer dette forholdet. I en layout må alle noder ligge på kun et nivå i strukturen, men sykliske lenker gjør at noen noder kan ligge på flere nivåer. I figuren ser vi at B ligger på nivå 2. Dette er logisk, siden B er barn av R, og R ligger på nivå 1. Men da burde B også ligge på nivå 4, siden B også er barn av A. Dette går ikke å vise på en todimensjonal layout, og en traversering ville eventuelt gått i sirkler. Vi har løst dette problemet ved å skille mellom **primære** og **sekundære noder**. For å forklare kan vi bruke Figur 6-6 som eksempel:

Ved oppstart lager PageMaker en tom matrise. Så lager PageMaker en Page (node) av startfilen R. PageMaker ser ut i fra indeksfilene at R lenker til F og B. Disse er R sine barn, og deres unike dokumentnummer blir lagret i R. Så blir R lagret i matrisen. Det

lages nye Page for F og B. F og B lagrer dokumentnummeret til R som deres far, og dokumentnummeret til deres barn igjen. Når vi treffer på node A, vil F legges til som far. B blir lagt til som barn av A, men PageMaker vil ikke traversere fra B igjen, fordi B allerede ligger på primært nivå 2. Rent programmeringsmessig blir det dette gjort ved at det primære nivået blir bestemt når en Page blir lagt til i matrisen. Hvis andre noder ser at deres barn allerede ligger i matrisen, vet de at en annen far har gjort krav på noden som sitt primære barn. Selv kan de bare ha dette barnet som et sekundært barn.

Indekseringen skjer automatisk når brukeren logger seg inn på Opsys. Grunnen til dette er at indekseringen er ganske tung å prosessere i en nettleser. Dessuten vil en ferdig indeksering gjøre at den grafiske delen av Lupen starter kjapt og lett fordi den ikke trenger å beregne noen tunge matriser.

6.3.2 Dokumentinformasjon

For å finne eksterne lenker, bilder og animasjoner har vi laget et eget program **FileParser**. FileParser er en Java-applikasjon, og må kjøres direkte på tjeneren av webmaster. Den søker igjennom alle HTML dokumenter under dbase-katalogen etter HTML-attributter, og genererer 3 filer. Hvilken linje attributtene ligger på, gjenspeiler det unike dokumentnummeret. Teksten på den første linjen er attributtene til dokument nummer 1, og så videre. Tegnet ”|” skiller flere forekomster av attributter.

hyper.dat inneholder alle hyperlenker til dokumentene.
Eksempel: `|http://www.porthcurno.org.uk/refLibrary/Semaphore`

anim.dat inneholder alle animasjoner til dokumentene.
Eksempel: `|Anim`

images.dat inneholder alle bilder til dokumentene.
Eksempel: `|ledig_plass_liste.gif|aktuelle_tilfeller.gif`
(Hvis denne teksten ligger på linje 25 i filen, har dokument nummer 25 bildene ”Ledig_plass_liste.gif” og ”aktuelle_tilfeller.gif”)

6.4 Layout

6.4.1 Forstørring

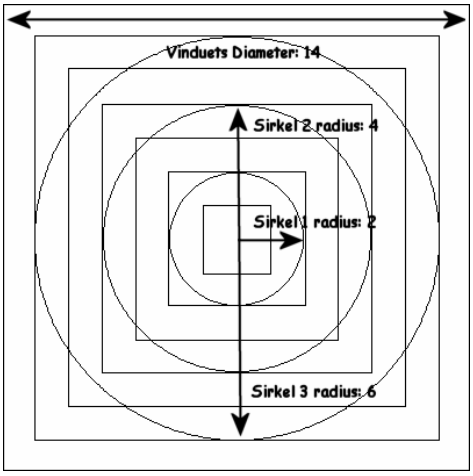
Når brukeren trykker på knappen for Lupen blir et vindu generert. Det skapes en klasse **EyeFrame** som er et tilfelle av Javas Frame-klasse. EyeFrame initialiserer så **GUI**, som er et tilfelle av Panel-klassen. EyeFrame legger så GUI oppå seg selv (Man kan se GUI i Figur 6-1 som det grønne feltet). All tegning skjer på GUI. Som vi lærte i prototypen var det svært viktig at GUI var helt firkantet for at matematikken bak layouten fungerte. Men brukeren må også kunne forstørre og forminske bildet. Ved vanlig forstørring av et vindu, kan brukeren selv bestemme høyde og bredde, noe som mest sannsynlig vil resultere i et rektangel. Vi tar derfor og adderer høyden og bredden, og deler summen på 2. Denne summen bruker vi da til å sette en gjennomsnittlig og firkantet, høyde og bredde på vinduet.

```
//Klasse GUI
private int MIN_SIZE = 400;
private int INIT_SIZE = 700;
private int MAX_SIZE = 1000;

private int getSquare() {
    Dimension d = new Dimension(getSize());
    int w = d.width;
    int h = d.height;
    int mean = (w + h) / 2;
    int return_value =  INIT_SIZE; //standard
    //beregner gjennomsnitt av brukerens valg
    if ( (mean > MIN_SIZE) && (mean < MAX_SIZE))
      return_value = mean;
    //valgt for liten størrelse, setter minimum
    if (mean <= MIN_SIZE)
      return_value = MIN_SIZE;
    //valgt for stor størrelse, setter maximum
    if (mean >= MAX_SIZE)
      return_value = MAX_SIZE;
    return return_value;
  }
```

Som vi ser i denne koden må brukeren holde seg innenfor en størrelse mellom 400 og 1000 piksler. Hvis han forstørrer vinduet innenfor denne rammen, vil metoden beregne gjennomsnittet av forstørringen horisontalt og vertikalt, og oppdatere GUI med den nye størrelsen. Det er ikke mulig å gå utover grensestørrelsene.

6.4.2 Plassering

Origo på GUI finnes ved å dele diameteren i to. Siden det skal vises 3 sirkler med noder, må det bestemmes en radius for hver av dem. Denne radiusen kan ikke gå til ytterkanten av vinduet. Dette er fordi nodenes posisjon blir bestemt av et par XY-koordinater, og disse koordinatene gjelder nodenes øverste venstre hjørne. En node på høyre side kan da risikere å havne utenfor vinduet. Vi løste dette problemet ved å dele GUI inn i 14 lengder, som vi ser i Figur 6-7. Vi brukte kun de innerste 12 lengdene, og dermed hadde vi en lengde til overs på hver side av firkanten. I tillegg er det like stor avstand mellom alle sirklene (2 lengder).



Figur 6-7 GUIs interne geometri

Som vi har sett har alle nodene informasjon om hvem de har som far, og hvem de har som barn. All denne informasjonen blir laget av PageMaker, og gitt videre til EyeFrame som en matrise. EyeFrame blir også gitt dokumentnummeret til startnoden.

EyeFrame har ansvaret for å kalkulere nodenes utgangsposisjon, og gir deretter nodene over til GUI for tegning. Kalkuleringen skjer ved hjelp av tre metoder, en for hver sirkel. Den overordende funksjonaliteten ved plasseringen er **sektordeling av sirkler**. Begynnelsen på en sektor er ved grad 0 (horisontal høyre side). Node A blir plassert i origo. Hvis A har 4 barn, blir sirkelen delt inn i 4 sektorer (360º/4 = 90). Disse sektorene blir så delt inn i nye sektorer, avhengig av disse nodenes barn igjen. Vi ser koden til den første sirkelen, nummeret til toppnoden blir gitt som parameter:

```
//plasserer noder i origo og på første sirkel.
private void setCircleOne(int startnr) {
  String str;
  Page p;
  Page par = (Page) pages.elementAt(startnr);
  dia = gui.getDia();
  //plasserer toppnode i origo
  par.setPos(dia/2, dia/2);
  par.setActive(true);
  gui.add(par);
  int x1 = dia/2;
  int y1 = dia/2;
  int x, y, n, nr;
  //beregning av radius
  int radius = (int) dia/7;
  Vector c = par.getChildren();
  n = c.size();
  //for alle toppnodes barn
  for (int i = 0; i < c.size(); i++) {
    x = (int)(radius * ( Math.cos((2*PI*i)/n))) + (dia/2);
    y = (int)(radius * ( Math.sin((2*PI*i)/n))) + (dia/2);
    str = c.elementAt(i).toString();
    nr = Integer.parseInt(str);
    p = (Page)pages.elementAt(nr);
    //plasser barn
    p.setPos(x, y);
    p.setActive(true);
    p.setOwner(par.getNumber());
    //beregn XY-koordinater til barnas lenker
    updateLines(par.locx,par.locy,x,y,par.getNumber(), p.getNumber());
    gui.add(p);
    //tving oppdatering av skjermbildet
    repaint();
  }
  //send barn til neste sirkel som foreldre
  setCircleTwo(c);
 }
```

Den neste sirkelen fungerer på nesten samme måte. Den tar inn barna fra forrige metode. Disse barna fungerer nå selv som far for sine barn igjen. For hver av disse fedrene blir barnas posisjon beregnet igjen. Utgangpunktet for ny tegning er nå ikke grad 0, men øverste grad på sektoren som faren ble tildelt. Som et resultat er ligningen forandret litt:

```
n = antall foreldre
N = antall barn til hver forelder
i = teller som teller opp til antall foreldre(n)
j = teller som teller opp barn til hver forelder(N)
x= (int)((radius*Math.cos(((2*Math.PI*N*i)+(2*Math.PI*i)-(Math.PI*N)-
(Math.PI)+(2*Math.PI*(j+1)))/(n*(N+1)))+(dia/2)));
y= (int)((radius*Math.sin(((2*Math.PI*N*i)+(2*Math.PI*i)-(Math.PI*N)-
(Math.PI)+(2*Math.PI*(j+1)))/(n*(N+1)))+(dia/2)));
```

Implementeringen av den siste sirkelen er ikke blitt fullført på en korrekt måte grunnet tidspress. Nodene tar ikke hensyn til sektorene til fedrene sine, men blir lagt ut på grad 0 igjen, på samme måte som i sirkel en. Dette medfører at nodene er forskjøvet en del i forhold til fedrene.

6.4.3 Tegning av noder og lenker

All egenkomponert tegning i Java benytter **Graphics** klassen. Graphics er en abstrakt klasse for alt grafikkbasert innhold. Når man tegner i Java, tegner man på Graphics. Alle grafiske komponenter i Java, det vil si de kan vises på skjerm, har en Graphics klasse. Vi kan for enkelhets skyld sammenligne Graphics med en tavle[32]. Oppdatering av grafikk skjer når vi kaller Graphics **update-metode**[33]. Denne metoden sikrer at komponentene selv oppdaterer seg, for eksempler når et bilde blir tildekket og så avslørt igjen av et annet vindu.

I Lupen tegner vi på GUIs Graphics klasse, og benytter dens update-metode. Ved oppstart av vinduet, blir alle noder tildelt XY-koordinater. Deretter blir nodene lagt til GUI for tegning:

```
//Klasse GUI

//metode for å legge til en node til GUI
public void add(Page r) {
    sprites.addElement(r);
    repaint();
}

//metode for tegning (svært forenklet)
public void update(Graphics g) {
  for (int i = 0; i < sprites.size(); i++) {
    Page r = (Page) sprites.elementAt(i);
    r.paint(g);
  }
}
```

Når GUI blir tegnet/oppdatert i update, sendes Graphics-objektet til alle tilfeller av Page som vises på skjermen. De kan så tegne seg selv på dette objektet. Etter at alle tilfeller av Page har tegnet på Graphics, blir Graphics til slutt tegnet på skjermen av GUI. Dette gjør at Page selv er ansvarlig for sin egen tegning, noe som vi lærte i prototypen.

I prototypen kom vi også over problemet med deling av lenker mellom nodene. Dette ble løst ved hjelp av en hashtabell, og multiplisering av dokumentnumrene. EyeFrame oppretter en tom hashtabell hver gang den beregner plasseringen til nodene. For hver to noder som deler en lenke, multipliserer EyeFrame deres unike dokumentnummer.

[32] Java Software Solutions 1998, Lewis &Loftus.
[33] Vi kan også benytte paint() direkte, men det er mer uvanlig. Ved bruk av paint() vil oppdatering skje tregere, og skjermen kan flimre. Med update kan vi benytte dobbelbuffering, en teknikk for å unngå flimring.

Denne summen blir nøkkel i hashtabellen. Innholdet er to par XY-koordinater, et par for hver av nodene:

Eksempel på en hashtabell.

NØKKEL	INNHOLD
2520 (node 28 ganger node 90)	(345, 234) & (567,456)
3213 (node 51 ganger node 63)	(342, 250) & (576, 530)

Dette innebærer at to noder kan dele en linje[34]. Når en node vil tegne linjene til sine barn, tar den barnas dokumentnummer (som ble tildelt noden av PageMaker), og ganger dette med sitt eget dokumentnummer. Resultatet er nøklene til hashtabellen, og dermed kan noden hente koordinatene til de linjene som den vil trenge. Under ser vi en forenklet versjon av Pages update-metode:

```
//Klasse Page
//forenklet metode for tegning
public void update(Graphics g){
  if (!children.isEmpty()){
    for (int i = 0; i < children.size(); i++){
      String str = children.elementAt(i).toString();
      temp = Integer.parseInt(str);
      //beregn nøkkel
      sum = number*temp;
      String key = Integer.toString(sum);
      //hvis nøkkel eksisterer
      if (hash.containsKey(key)){
        Line l = (Line) hash.get(key);
        Page p = (Page) pages.elementAt(temp);
          //barnet vises på skjermen
        if (p.getActive()) {
          // tegn alle linjer
          if (showAll) {
             //tegn linjene
          }
          //tegn kun ikke-sykliske lenker
          }else {
            //hvis barnet har meg som primær far
            if (number == p.getOwner()){
             //tegn linjene
            }
  //tegn resten av noden…
```

Hvilke linjer som noden skal tegne er avhengig av to forutsetninger: Om brukeren vil se sykliske lenker, og om barnnoden er synlig på skjermen. Hvis brukeren bare vil se ikke-

[34] Hashtabellen er synkronisert. Det vil si at bare en prosess kan hente et element i tabellen, selv om en annen prosess vil hente samme element samtidlig. I slike tilfeller vil en lenke bli tegnet to ganger, først av den første prosessen, så av den andre prosessen.. Denne ulempen er liten, og gir ingen praktiske problemer.

sykliske lenker, vil noden sjekke om barnet har et primært forhold til seg. Om barnet har det, vil lenken tegnes. Hvis ikke blir ikke lenken tegnet.

Hvis barnnoden ikke er synlig på skjermen, sykliske lenker eller ikke, vil ikke noden tegne lenken (det er liten vits å tegne en lenke som fører ut av vinduet).

6.5 Historien

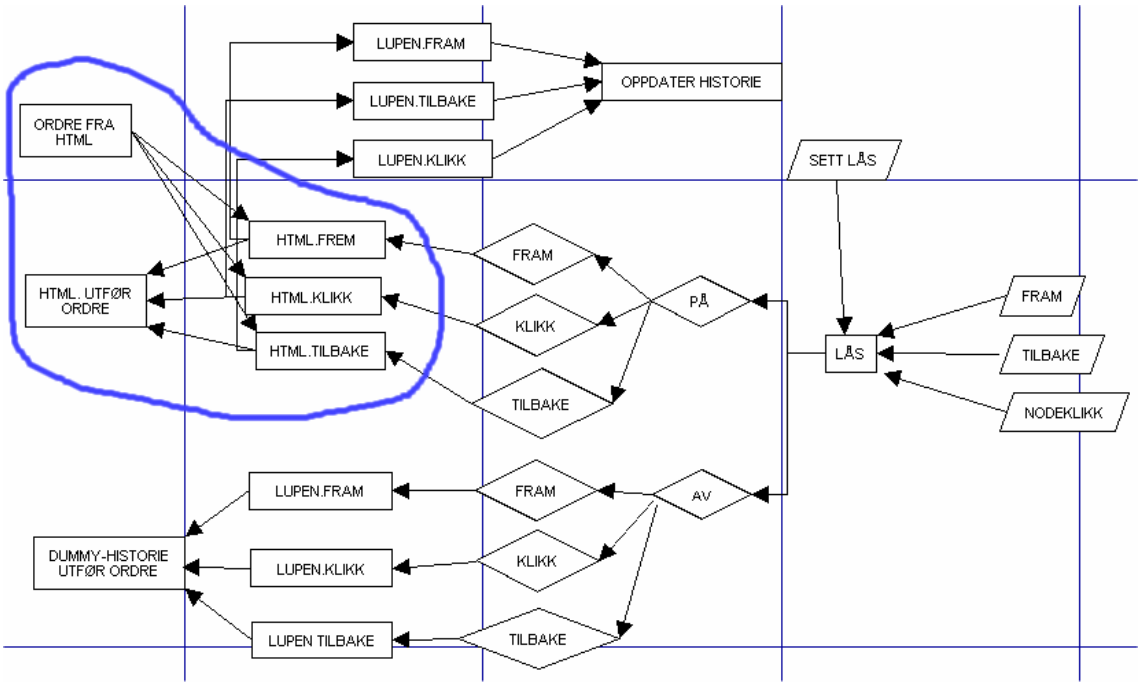
6.5.1 Navigering

Opsys har knapper for å gå fram og tilbake, på samme måte som en nettleser har. Denne arkitekturen benytter JavaScript. Knappene er deklarert i menyrammen, og kaller opp nettleserens egendefinerte tilbake-metode eller fram-metode:

```
//kode fra menyrammen
<A HREF="javascript:tilbake()" onMouseOver="window.status=''; return true;">
<IMG Border="0" SRC="dbase/img/icons/back.gif"></A>

//Javascript
function tilbake() {
  //kobling til nettleserens tilbake-metode
  parent.frames[1].history.back();
}
```

Dette fungerer bare en vei: fra menyrammen til nettleseren. For å få et nettkart som ikke er tilstandsløst, måtte vi sørge for at klikk på lenker og navigeringsknapper i Opsys, også gir beskjed om dette til Lupen. I tillegg må Lupen kopiere historiefunksjonaliteten som er innebygd i nettleseren, da nettleserens historieliste ikke er tilgjengelig for noen kall fra Java[35]. Vår metode måtte også ta hensyn til Lås-funksjonaliteten, det vil si at knyttingen til nettleseren må kunne slåes av og på, og samtidig koordinere dette med nettleseren. Utviklingen av alle disse metodene var svært krevende, og innebar en meget systematisk framgangsmåte. Hvordan historielisten fungerte i nettleseren var også ukjent, så vi måtte prøve oss frem.

[35] Den var opprinnelig tilgjengelig i tidligere nettlesere.



Figur 6-8 Flowchart av historien. Blå sirkel markerer menyrammen.

Som vi ser i Figur 6-8, blir navigasjonskallene fra Lupen først sendt direkte til menyrammen hvis Låsen er slått på. Menyrammen behandler disse, og sender så kallene tilbake igjen til Lupen. Først da vil Lupen faktisk reagere internt på kallene. Denne måten å sende kallene i ring på, på sørger for at vi maksimerer de allerede eksisterende metodene i Opsys. Hvis Låsen er slått av, vil alle kallene behandles internt i Lupen. De vil da ikke gå til menyrammen for prosessering.

6.5.2 Intern Historie

Den interne historien er en kopi av nettleserens historieliste. Vi bygger opp en liste av dokumentnummer, og setter en variabel til å peke på brukerens posisjon. For hver navigeringsordre, vil den interne historien oppdatere den nye posisjonen, og fjerne eller legge til dokumentnummer.

De to viktigste metodene i historie-klassen er *orderToJAVA* og *routedOrders*.

OrderToJava blir kalt opp fra HTML-siden. Kallene kan komme som et direkte kall fra menyrammen, eller som et kall som opprinnelig kom fra Lupen, men ble sendt rundt til menyrammen først. Denne runddansen skjer når låsen er slått på.

routedOrders blir kalt opp fra Lupen selv. Dette gjør at HTML-siden ikke vet noe om hva som skjer. Dette skjer når låsen er slått av.

```
//Klasse History

//variabel som indikerer posisjonen til brukeren i historielisten
private int pointer = 0;
//indikerer om det er mulig å gå fremover
private boolean fpos = false;
//liste som inneholder historien. Hvert element er et dokumentnummer
private Vector list;

//constructor. Tar inn et dokumentnummer for startdokumentet
public History(int s) {
  makeHash();
  parseNr(s);
}

//Bestemmer hvilke metoder som skal kalles. Blir kalt opp
//av Javascript i Menyrammen
public void orderToJAVA(String s){
  if (s.equals("tilbake")) {
    goBack();
  } else if (s.equals("frem")) {
    goForward();
  } else {
    Integer x = new Integer(s);
    int nr = Integer.parseInt(s);
    parseNr(s);
  }
}

//samme som orderToJava, men blir kalt internt fra Java
public void routedOrders(String s) {
  if (s.equals("tilbake")) {
    dummy.goBack();
  } else if (s.equals("frem")) {
    dummy.goForward();
  } else {
    Integer x = new Integer(s);
    int nr = Integer.parseInt(s);
    dummy.parseNr(s);
  }
}

//ordre om å gå et steg tilbake
public void goBack(){
  //kan ikke gå tilbake. Listen inneholder ikke nok noder
  if (pointer < 1){
  }else {
    pointer = pointer - 1;
    String str = (String) list.elementAt(pointer);
    Integer I = new Integer(str);
    if (lock) {
      //kall opp ExecDoc i menyrammen
      launcher.click(I.parseInt(str));
    }
```

```
  }
  //sett variabel som indikerer om det er mulig å gå fram nå
  forwardpossible();
}

public void goForward(){
  //allerede så langt fram som det er mulig å gå
  if (pointer > list.size()-2) {
  }else{
      pointer = pointer + 1;
      String str = (String)list.elementAt(pointer);
      Integer I = new Integer(str);
      if (lock) {
        //kall opp ExecDoc i menyrammen
        launcher.click(I.parseInt(str));
      }
  }
  forwardpossible();
}

//er det mulig å gå et steg fram nå
private boolean forwardpossible(){
  fpos = pointer < list.size()-1;
  return fpos;
}

//bygger opp listen til historien
private void parseNr(String s){
  int nr = Integer.parseInt(s);
  Page p = (Page) pages.elementAt(nr);
  //listen er tom. Legg til siden
  if (list.isEmpty()) {
    list.addElement(s); //added startnr
    if (lock){
      //kall opp ExecDoc i menyrammen
      launcher.click(Integer.parseInt(s));
    }
  }
  else { //Liste er ikke tom
    String in = (String) list.elementAt(list.size()-1);
    if (in.equals(s)){ //Lik forrige doknr, bare ignorer ordre
    }else{ //ikke lik forrige
      if (fpos) { //er mulig å gå frem
        String old = (String)list.elementAt(pointer+1);
        //dok ligger allerede på denne plassen. Trenger ikke bygge opp listen
        if (s.equals(old)){
          nr = Integer.parseInt(s);
          if (lock) {
            launcher.click(nr);
          }
          pointer++;
        }
        //bygg opp ny sti i listen
        else {
          for (int x = pointer+1; x < list.size(); x++) {
            list.removeElementAt(pointer+1);
            x--;
          }
          list.addElement(s);
          nr = Integer.parseInt(s);
          if (lock){
            launcher.click(nr);
          }
          //oppdater pointer til den nye plassen
          pointer = list.lastIndexOf(s);
        }
      }
      else { //er ikke mulig å gå fram, bare legg til
```

```
            list.addElement(s);
            nr = Integer.parseInt(s);
            if (lock){
              launcher.click(nr);
            }
            //oppdater pointer til ny plass
            pointer = list.lastIndexOf(s);
          }
      }
    }
    pointer = list.lastIndexOf(s); //plassen hvor det siste elementet ble added
    forwardpossible();
    //metode for å oppdatere brukerens historie på tjeneren
    parseHash(nr);
}
```

Når brukeren slår av låsen, blir historiens liste lagret sammen med brukerens posisjon i denne. Deretter vil kontrollen gå over til en ny historieliste **DummyHistory**. Så lenge låsen er av, vil alle navigeringsvalg kun oppdatere DummyHistory. Hvis brukeren så slår på låsen igjen, vil den vanlige historielisten gjenopptas. Posisjonen vil da gå tilbake til det siste dokumentet hvor brukeren hadde låsen slått på.

6.5.3 Lesing og skriving med skript.

Hver bruker har sin egen historie. Denne lagres på brukernes hjemmekatalog som en vanlig tekstfil. Tekstfilen har formatet:

```
Dokumentnummer::Dato|Klokkeslett#Antall ganger besøkt.
```

- Eksempel: 66::23.09.2003|02:53#8

Denne filen leses inn ved oppstart, og oppdateres kontinuerlig under brukerens sesjon. Lesingen og skrivingen skjer ved hjelp av en QueryString og Perl. Historieklassen i Lupen forbereder en string med all informasjon som Perl trenger i formatet:

```
URL?kommando=dokumentnummer::Dato|Klokkeslett=brukernavn
```

- Eksempel: http://thatcher/history.pl?update=84::20.10.2003|08:57=trondal

Kommandoer kan være ”oppdater”, ”legg til” og ”les”. Perl vil returnere respektive ”OK” eller ”Error”.

```
Klasse history i Java
//Oppdatering av brukerens historie (forenklet)
```

```
private void setUserHistory(int nr){
    Date d = new Date();
    try {
     String update = scriptURL+"?update="+nr+"::"+date+"="+user;
     URL url = new URL(update);
     uc = url.openConnection();
     uc.setAllowUserInteraction(true);
     uc.setUseCaches(false);
     uc.connect();
     br = new BufferedReader(new InputStreamReader(uc.getInputStream()));
     String line = br.readLine();
     br.close();
   } catch(IOException ee) { ee.printStackTrace();}
  }

//PERL, history.pl
use CGI::Carp qw(fatalsToBrowser carpout);
print "Content-type:text/plain\n\n";

//ta inn Query String som miljøvariabel
$query_string = $ENV{'QUERY_STRING'};
//konverter til vanlig tekst
$query_string =~ s/%([\dA-Fa-f][\dA-Fa-f])/pack (
                         "C", hex ($1))/eg;
$query_string =~ s/\+/ /g;
//split string inn i kommandoer osv.
($key, $value, $user) = split(/=/, $query_string);
$path = "d:/http/emner/mnfit222/opsys/users/" . $user . "/history.dat";
…
//avgjør hvilken rutine som skal kjøres
} if ($key eq 'update'){
    &update($value);
...

sub update {
  use Tie::File; //modul for knytting av en fil til en matrise
  my ($value) = @_;
  tie @array, 'Tie::File', $path or die "Error";
  ($nr, $date) = split(/::/, $value);
  $int;
  $count = 0;
  for (@array) {  #traverser filen
       my $line = $array[$count];
    ($old_nr, $old_date) = split(/::/, $line);
       ($old_date, $index) = split(/\#/, $old_date);
       $index = $index + 1;

       if ($old_nr eq $nr) {
      $value = $value . "#" . $index;
         splice @array, $count, 1, $value;
       }
       $count++;
  }
  untie @array;
  print "OK";
}
```

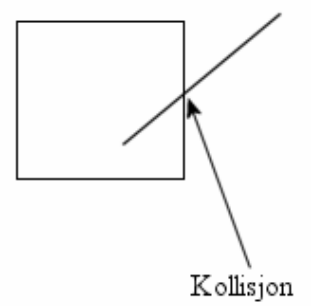
Perlskriptet forutsetter en Perl-versjon 5.0.8 eller nyere. Dette er på grunn av bruken av Tie::File modulen i skriptet.

6.6 Avansert tegning

For at lenkene ikke skal tegnes over selve nodene, må man beregne når overlappingen mellom streken (lenken) og firkanten(noden) skjer. Vi benyttet Javas kollisjonsmetode (getIntersectionPoint) for å oppnå dette:

```
public Point getIntersectionPoint(Line2D.Double line, java.awt.Rectangle
shape)
```

Metoden tar inn en linje og et rektangel som parameter. Så traverserer den alle piksler til rektangelet, og avgjør om noen punkter på linjen vil overlappe noen av disse. Resultatet vil bli et punkt som er kollisjonspunktet.



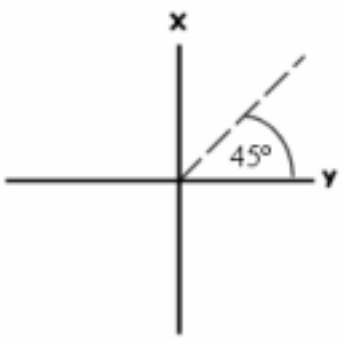
Figur 6-9 Kollisjon

I tillegg må det tegnes en pil for å indikerer retningen på lenken. Vi tar kollisjonspunktet, og tegner en polygon ut i fra dette:

```
//Klasse Page

public void update(Graphics g){
..
g.drawPolygon(getArrow(p2.x, p2.y, p1.x, p1.y, 7, 5, 0.2));
..

private Polygon getArrow(int x1, int y1, int x2, int y2, int headsize,
                         int difference, double factor) {
   int[] crosslinebase = getArrowHeadLine(x1, y1, x2, y2, headsize);
   int[] headbase = getArrowHeadLine(x1, y1, x2, y2, headsize - difference);
   int[] crossline = getArrowHeadCrossLine(crosslinebase[0], crosslinebase[1],
                                           x2, y2, factor);
   Polygon head = new Polygon();
   head.addPoint(headbase[0], headbase[1]);
   head.addPoint(crossline[0], crossline[1]);
   head.addPoint(x2, y2);
   head.addPoint(crossline[2], crossline[3]);
   head.addPoint(headbase[0], headbase[1]);
   head.addPoint(x1, y1);
   return head;
 }

private int[] getArrowHeadLine(int xsource, int ysource, int xdest, int ydest,
                               int distance) {
   int[] arrowhead = new int[2];
   int headsize = distance;
   double stretchfactor = 0;
```

```
   stretchfactor = 1 - (headsize / (Math.sqrt( ( (xdest - xsource) * (xdest -
xsource)) + ( (ydest - ysource) * (ydest - ysource)))));
   arrowhead[0] = (int) (stretchfactor * (xdest - xsource)) + xsource;
   arrowhead[1] = (int) (stretchfactor * (ydest - ysource)) + ysource;
   return arrowhead;
 }

private int[] getArrowHeadCrossLine(int x1, int x2, int b1, int b2, double
factor) {
   int[] crossline = new int[4];
   int x_dest = (int) ( ( (b1 - x1) * factor) + x1);
   int y_dest = (int) ( ( (b2 - x2) * factor) + x2);
   crossline[0] = (int) ( (x1 + x2 - y_dest));
   crossline[1] = (int) ( (x2 + x_dest - x1));
   crossline[2] = crossline[0] + (x1 - crossline[0]) * 2;
   crossline[3] = crossline[1] + (x2 - crossline[1]) * 2;
   return crossline;
}
```

Selve animeringen av nodene blir styrt ved hjelp av tråder (Threads) i GUI. Ved hver klikk for animering, vil en ny tråd startes i GUI, og oppdatere Page med de nye XY-koordinatene. Koordinatene blir oppdatert i forhold til en tidsperiode. Vi har valgt 20 millisekunder som en periode som både er rask i animeringen, og effektiv for en nettleser å tegne.



Figur 6-10 Stigninggrader i animasjon

Problemet med XY-koordinater i en slik animasjon er at det ikke alltid er like stor forandring på X som på Y. Hvis for eksempel stigningstallet er over 45 grader, vil man få flere X-koordinater enn Y-koordinater ved bruk av ”vanlige” matematiske metoder. Dette vil da medføre at en node som har en høy stigningsgrad, vil animere seg fortere tilbake til origo, enn en node som har en lavere stigningsgrad. Vi måtte finne en metode for å få like mange koordinater uansett stigningstall. Vi benyttet derfor en spesiell DDA-algoritme fra nettsidene til CodeGuru[36]:

```
public void run() {
    //Digital Differential Analyzer - improved over symmetry
```

[36] www.codeguru.com

```
    Thread thisThread = Thread.currentThread();
    Page p = focus;
    //p.setDraggable(false);
    int x0 = p.locx; //der man er
    int y0 = p.locy; //der man er
    int x1 = p.org_locx; //dit man vil
    int y1 = p.org_locy; //dit man vil
    int dy = y1 - y0;
    int dx = x1 - x0;
    float t = (float) 0.5; // offset for å runde av
    if (Math.abs(dx) > Math.abs(dy)) { // stignigstall er mindre enn 1
      float m = (float) dy / (float) dx; // beregn slope
      t += y0;
      dx = (dx < 0) ? -1 : 1;
      m *= dx;
      while (x0 != x1) {
        x0 += dx; // gå til neste x verdi
        t += m; // legg til slope to y verdi
        p.updatePos(x0, (int) t);
        rotation = rotation + 1;
        //roter hjul I origo
        wheel.rotateWheel( -rotation);
        repaint();
        try {
          thisThread.sleep(REFRESH_RATE);
        }
        catch (InterruptedException exc) {}
        ;
      }
    }
    else { // stigningstall er 1
      float m = (float) dx / (float) dy; // beregn slope
      t += x0;
      dy = (dy < 0) ? -1 : 1;
      m *= dy;
      while (y0 != y1) {
        y0 += dy; // gå til neste y verdi
        t += m; // legg til slope til x verdi
        p.updatePos( (int) t, y0);
        rotation = rotation + 1;
        //roter hjul I origo
        wheel.rotateWheel( -rotation);
        repaint();
        try {
          thisThread.sleep(REFRESH_RATE);
        }
        catch (InterruptedException exc) {}
        ;
      }
    }
  }
```

7 Drøfting og forslag til videre arbeid

7.1 Innledning

I dette kapittelet skal vi se på Lupens svakheter, og gi forslag til løsninger som kan bli benyttet ved utvikling av liknende program i fremtiden. I tillegg vil vi se på tjenester som også kan bli implementert i fremtiden. Til slutt vil vi se hvordan mangelen på standarder i dagens Internett gir store utfordringer til nettkart generelt.

Lupen er i dag fullt funksjonell, og har vært tilgjengelig for studentene på faget MNFIT222 Operativsystemer hele høsten 2003. Vi har utviklet et statistikkprogram for å analysere brukernes bruk av Opsys. Dessverre har det ikke vært tid til å utvide Lupen med samme statistikk. Derfor har vi ingen faste holdepunkter for å diskutere Lupens brukbarhet, og om programmet faktisk var til hjelp for brukerne. For selve bruken av Opsys høsten 2003 har vi likevel registrert følgende informasjon:

- Opsys har 1008 sider
- Det var 87 registrerte brukere totalt.
- 65 av disse har benyttet Opsys.
- De 65 har gjennomsnittlig besøkt 348 sider hver.

Dette viser at Opsys som system er relativt mye brukt. En kognitiv vurdering av selve Lupen vil måtte basere seg på en statistisk undersøkelse hvor brukerne måtte finne fram til en side med eller uten bruk av Lupen. Dette har vi som sagt ikke hatt tid til. Likevel har vi vist at utviklingen av et dynamisk nettkart ligger innenfor rekkevidde for de forfattere og Webmastere som vil benytte et slikt verktøy på deres egne nettsider.

7.2 Svakheter i Lupen

7.2.1 Ikke direkte overførbart program

Opsys er et filbasert system, til forskjell til et databasebasert system. Dette medfører at arkitekturen i Opsys er svært egendefinert. Vi dro fordel av dette, i og med at vi slapp å

utvikle egne URL-baserte edderkopper for å bygge opp nettverksstrukturen. Men vår løsning ble da bare en videreutvikling av en allerede vanskelig og utdatert løsing. Denne problemstillingen var klar for oss tidlig i utviklingen, men arbeidet med å utvikle en URL-basert edderkopp ble sett på som for stort.

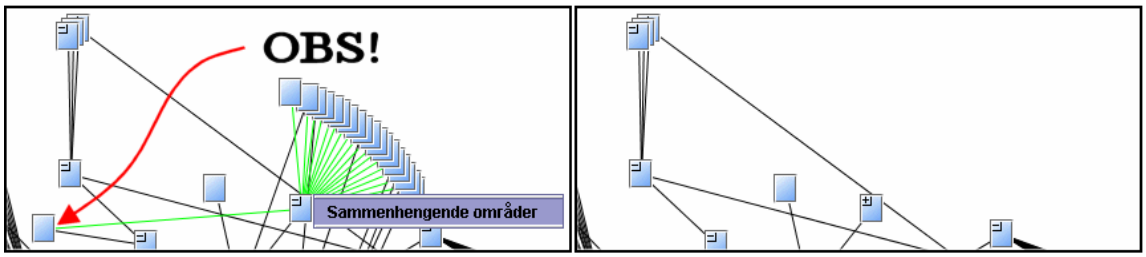
Den egendefinerte arkitekturen gjør at Opsys ikke kan benyttes på andre nettsteder uten videre. Det hadde vært ønskelig at Opsys og Lupen kunne blitt installert på andre nettsteder som et vanlig eksekverbart program. Man kunne da ved installeringen ha skrevet inn parametere som viste hypermediebase, start-URL og så videre.

7.2.2 Misting av sammenheng

Som vi nevnte i 5.5.3 mister brukeren all informasjon i den forrige noden, når han klikker på en node. Lupen viser kun alle lenker som går ut i fra den noden som til enhver tid er i origo (Figur 5-3). Lenken som førte brukeren dit, blir borte. Dette er forvirrende, da brukeren hele tiden blir presentert for forskjellige skjermbilder ved hvert klikk. Under utviklingen ble en implementering som tok hensyn til dette, utsatt på grunn av vanskeligheter med matematikken i layouten og selve logikken bak. Arbeidet med dette ble ikke gjenopptatt på grunn av tidspress.

7.2.3 Klumping av noder

Som vi ser i Figur 6-1 ligger noen av nodene i klumper på sirklene. Dette er fordi noen noder har mange barn. Det er ikke plass til å tegne alle sammen med ”luft imellom” på den tilgjengelige plassen. Plassproblemer er noe som alle dynamiske nettkart strever med. En tidligere prototype av Lupen prøvde å omgå dette problemet ved få nodene til å benytte **progressiv avsløring**. Noden kunne da fjerne sine egne barn fra skjermen. Men som vi kan se i Figur 7-1 ble da noder som har en syklisk lenke til en fjernet node også fjernet. Problemet er rekursivt: Hvis disse barna igjen har sykliske lenker, vil de også bli fjernet. Faktisk kan man risikere at node A fjerner node B som fjerner node C som fjerner node A igjen!



Figur 7-1 Eksempel på sammentrekking av en node

En løsning på dette kan være at nodene ikke fjerner barn som har sykliske lenker. Men uansett vil ikke den sykliske lenken vises. Hvor stort problem dette er, er uvisst. Vi benytter jo i dag en layout som ikke viser lenker til noder utenfor 3 nivåer, noe som er relativt likt.

7.3 Forbedringsforslag til Lupen

7.3.1 URLbasert edderkopp

Innhentingen av nettverksstrukturen bør være basert på URL og http-protokollen. Dette ville gjort Opsys og Lupen mer overførbar til andre nettsteder. Mappucino bruker denne metoden. Man taster kun inn URL-adressen til toppnoden, og traverseringen skjer i henhold til http-protokollen.

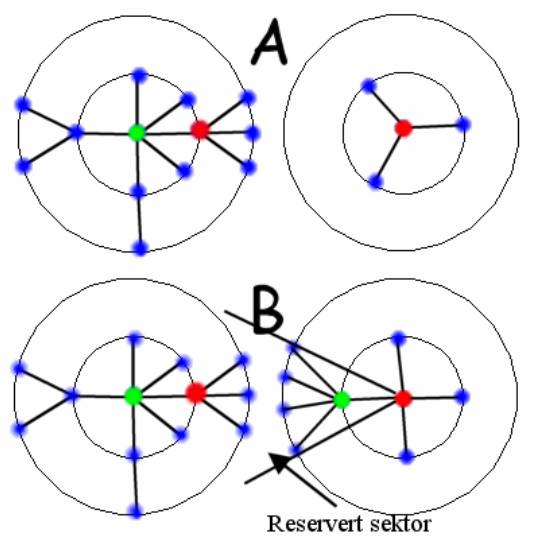
7.3.2 Bevaring av orientering

Lupen burde vist hvordan brukeren kom fram til en node. Den gjeldende implementeringen viser ikke dette forholdet. Vi kan se dette i Figur 7-2, hvor Lupen benytter tilfelle A.

I Lupen har hver node dokumentnummeret til:

- Fedrene. Alle noder som lenker til noden
- Barna. Alle noder som noden selv lenker til.

Dette er faktisk all den informasjonen som nodene trenger for å implementere en visning av den tidligere noden (som førte til den nye plasseringen). Den nåværende implementeringen har en metode for å traversere **nedover** nettverksstrukturen; Vi tegner startnoden i origo, alle barna i sirkel en, deres barn i sirkel 2 og så videre.

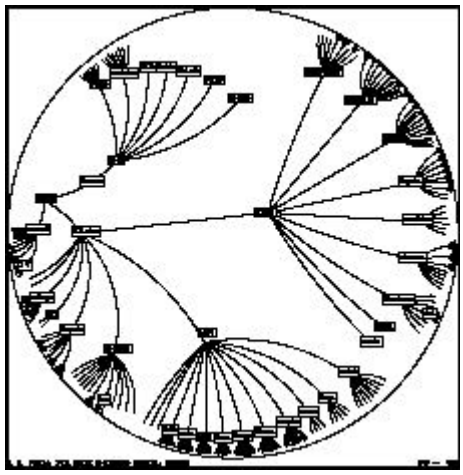


Figur 7-2 A: Mister tidligere node. B: Bevarer tidligere node

En implementering av tilfelle B ville forutsatt en metode for å traversere **oppover** igjennom nettverksstrukturen. Samtidlig måtte inndelingen av sektorene i sirkelen ta hensyn til at en sektor måtte bli **reservert** for lenken som førte brukeren til den gjeldende noden. Nodene i den reserverte sektoren måtte da bli tegnet på motsatt måte, med barnet (GRØNN) i den første sirkelen, og barnets foreldre i neste sirkel og så videre.

Man kunne også utviklet en mer sofistikert visning av nodene. Hvis man utviklet en layout som utnyttet plassen bedre, for eksempel ved å vektlegge nodene basert på antall lenker som går ut i fra hver node. En node med mange barn kunne da få større plass på skjermen, enn en node med få barn. Dette ville antakelig medført en visning av lenker som går over eller under nodene. Mulighetene er mange, men vi vil likevel hevde at en hyperboolsk layout ville vært mest ønskelig.

7.3.3 Animert hyperboolsk layout



Figur 7-3 Hyperboolsk layout

En hyperboolsk layout har den fordelen at svært mange noder kan vises på en relativt liten plass. De nodene som ligger lengst ute på kanten av horisonten er da meget små, og vil ikke gi noen meningsfull informasjon direkte. Men det ville likevel indikert eksistensen av disse nodene, og gitt brukeren en følelse av å se hele nettverket på en gang. Den hyperboolske layouten kunne også støtte animering, slik at brukeren kunne trekke og dra i ”kulen”. Denne animeringen ville heller ikke gjort at den tidligere fokuserte noden ville forsvinne, men bare blitt mindre.

Layouten har også noder som er klumpet sammen. Men det er da bare for brukeren å trekke disse nodene mot sentrum av sirkelen. Nodene vil da bli forstørret, og gradvis ville det blitt mer plass mellom nodene. Problemet med klumping av noder blir delvis løst av denne funksjonaliteten. Brukeren mister heller ikke orienteringen så lett, siden ingenting forsvinner med en gang, men gradvis blir mindre og mindre jo lengre vekk man kommer.

7.3.4 Personlige stier

Opsys har igjennom NavigasjonsAppleten støtte for forfatter- og egendefinerte stier. Det hadde vært ønskelig å videreføre dette i Lupen. Dette kunne blitt gjennomført med en applet til applet-kommunikasjon. Begge appleter baserer seg på Opsys’ notasjon om unike dokumentnummer, så en implementering av stier i Lupen ville ikke vært for

vanskelig. I den eksisterende arkitekturen lagrer NavigasjonsAppleten stiene i en egen fil som får navnet etter stien.

Stien ”Forelesning 1” blir lagret i filen ”Forelesning 1.path”. Innholdet følger formatet:

```
Dokumentnummer=dokumenttittel
```

Eksempel:

```
289=Oversikt over hele kurset
76=Intro/Oversikt, Kapittel 1
693=Bruker/anvendelsesprogram
553=Operativsystem
```

For at Lupen skal få denne informasjonen, hadde det vært nok at NavigasjonsAppleten hadde sendt melding om hvilken (om noen) sti som var valgt til Lupen. Lupen kunne da enten lest inn stien selv fra brukerens katalog, eller fått stien levert av NavigasjonsAppleten som et parameter.

Man kunne for eksempel uthevet sti-nodene i layouten, eller fått dem til å skifte farge. Man kunne også fjernet alle noder som ikke var i stien, og på denne måten kun vist den aktuelle stien i Lupen. Mulighetene er mange.

Stiene kunne også blitt knyttet til Lupens egne navigasjonsknapper. Ved innlasting av stiene, kunne den gjeldende historien blitt suspendert (satt på venting) og erstattet av stien. Man hadde bare lastet stien inn i historiens liste (stack), og satt pekeren på det aktuelle dokumentnummeret. Klikk på en lenke i Opsys som ikke var en del av stien måtte da blitt ignorert, hvis ikke ville listen blitt korrupt (inneholdt et nummer som ikke finnes i stien). Alle disse forandringene ville antakelig virket forvirrende på en bruker, så Lupen måtte da vise at den er i en **sti-modus**.

Sti-modusen kunne blitt vist ved hjelp av endring av farger, knapper og liknende. Det måtte også da være en tjeneste som kunne avslutte sti-modusen, og gå tilbake til den opprinnelige modusen Lupen var i.

7.3.5 Musen og nodene som objekter

Det hadde vært ønskelig å gjøre nodene til objekter i Lupen. Med dette mener vi at noden er selve dokumentet, ikke bare en representasjon av dokumentet. Analogien er den samme som i Windows OLE-teknikk.

For eksempel hvis man høyreklikket på en node, kunne man valgt ”skriv ut”, på samme måte som vanlige Windows-programmer. Forfatter kunne også hatt et valg om å editere dokumenter direkte i Lupen. Han kunne åpnet et dokument i for eksempel Word eller FrontPage ved å høyreklikke på en node. Forfatter kunne også trukket lenker imellom noder i Lupen, og dermed skapt reelle lenker mellom dokumentene. Dette kunne blitt utvidet til bygging av stier, fjerning av noder og så videre.

De tekniske implikasjonene bak dette er uklare. Vi skal ikke drøfte en teknisk framgangsmåte for å utvikle dette, da det er et studie i seg selv.

7.3.6 Visning av nodeattributter

Lupen tegner alle noder likt, uansett innhold. For å få mer informasjon om en node, må man høyreklikke på den, og velge dokumentets tittel i den påfølgende menyen. De første prototypene skilte mellom noder som hadde eksterne hyperlenker og de som ikke hadde det. En del av grunnen til dette var at Page-klassen til nodene utgikk fra Swings JButton-klasse. JButton hadde støtte for ikoner. De senere versjonene av Lupen brukte AWT, og all tegning var egendefinert. Dermed ble også tegning av forskjellige ikoner nedprioritert, og til slutt utelatt på grunn av tidspress.

I retrospekt burde brukeren i alle fall fått et valg om visning av forskjellig nodeattributter på selve noden. Dette vil gi brukeren muligheten for å skille mellom nodene: ”det var den noden nederst ved siden av hyperlenke-noden” Dette er relativ enkelt å implementere, siden nodene i dag har denne informasjonen innebygd som variabler i Page-klassen. Det er naturligvis ikke mulig å vise spesielt mye informasjon på nodene, da de er så små. Men eksempler på hva som kunne blitt valgt av brukeren kunne være:

- Eksterne hyperlenker
- Animasjoner

- Forekomst av bilder

Det ville antakelig bare være hensiktsmessig å bare vise et av disse attributtene samtidlig. Hvis ikke ville tittelen til noden bli skjult. I Figur 7-4 ser vi et tenkt eksempel.

Figur 7-4 Indikering av ekstern hyperlenke

Man kunne også gjort ”viktige” noder større enn de andre nodene. Forfatter kunne ha indikert dette i metadata. Opsys ville ha parset denne informasjonen når indeksfilene ble bygd med skriptet lagix.pl, og skapt en ny fil ”viktig.dat”. Her kunne de viktige nodenes dokumentnummer blitt lagret, og Lupen hadde lest denne filen ved oppstart. Størrelsen kunne også vært avhengig av størrelsen på dokumentet.

7.3.7 Visning av ”tapte lenker”

Slik Lupen er nå, vises alle lenker mellom alle de nodene som eksisterer på skjermen. Hvis ikke begge nodene blir vist samtidlig, vil heller ikke lenken vises. Dette er ikke tilfredsstillende, da det ikke gjenspeiler det faktiske nettverket. I Opsys finnes det flere slike lenker. Det er ikke ønskelig at forfatter skal ta hensyn til dette ved utvikling av innholdet; ekte hypertekst forutsetter at det kan være lenker mellom alle noder.

I utviklingsfasen tenkte vi på å indikere slike ”tapte” lenker med egne symboler på ikonene. Hvis brukeren trykte på symbolet, ville en meny vise tittelen på de nodene som den tapte lenken fører til. Tegning av selve lenken var ikke aktuelt, da det bare ville blitt en strek som førte ut i fra skjermen. Implementeringen av dette er ganske lett, men ble utelatt på grunn av tidspress.

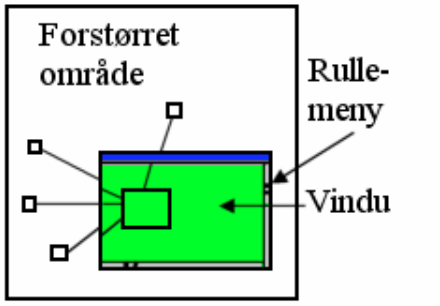
Figur 7-5 Plusstegnet indikerer ”tapte lenker”

7.3.8 Skille mellom typer av lenker

Lupen skiller mellom sykliske lenker og ikke-sykliske lenker. Denne funksjonaliteten kunne blitt utvidet til å vise forskjellige farger på de forskjellige lenkene.

7.3.9 Utvidelse av forstørrelsestjenesten

Det vinduet som forstørres har per i dag ingen interaktivitet, som nevnt i 6.2.7. Det hadde vært ønskelig at brukerne kunne klikket på disse nodene, på samme måte som i hovedvinduet i Lupen. Brukerne kunne da også rulle opp og ned på det samme vinduet, og dermed vise andre områder enn akkurat det som ble valgt. Analogien er den samme som ved bruk av mikrofilm på en prosjektør.

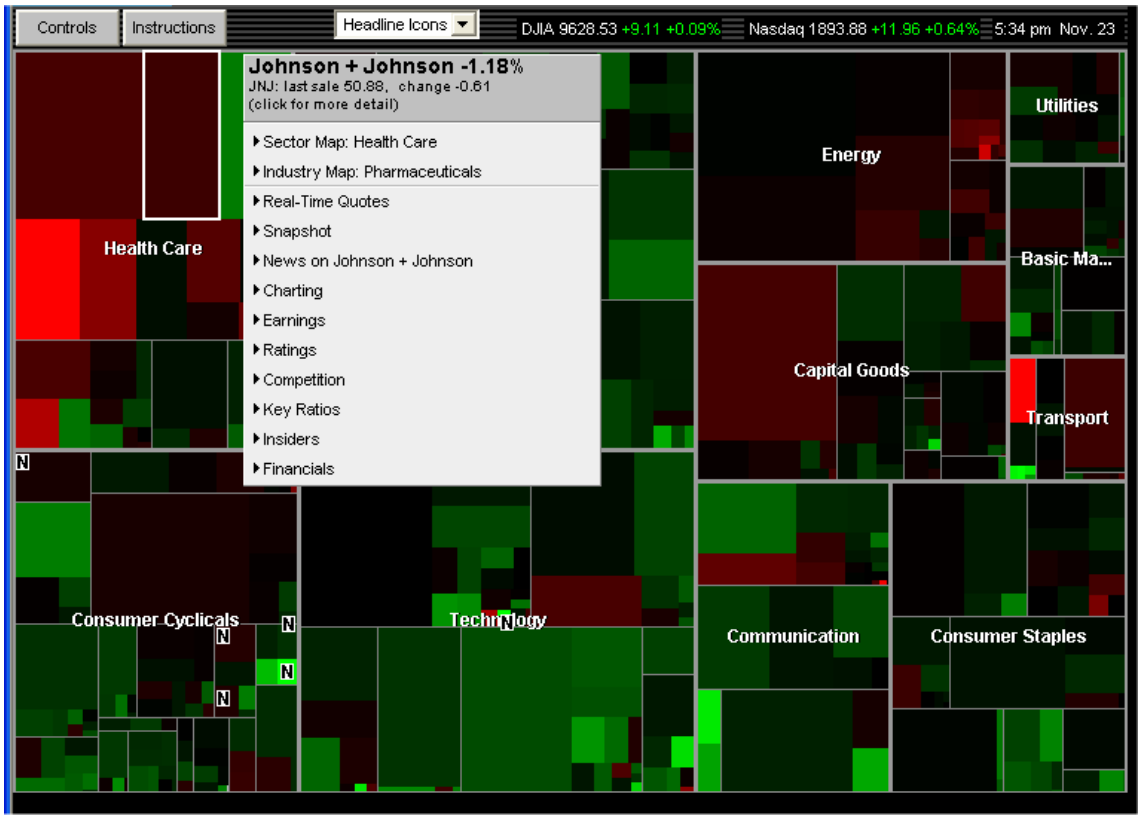


Figur 7-6 Forstørring av hele Lupen

I dag fungerer denne tjenesten ved at en søkemetode søker etter alle ikoner som befinner seg i det valgte området. Bredden og høyden til det valgte området blir forstørret 3 ganger, og et nytt vindu blir skapt med de forstørrede målene. Den relative posisjonen til nodene blir også beregnet ut i fra det valgte området, og forstørret 3 ganger. En kopi av nodene blir så plassert på det nye vinduet, men nå er nodene like brede som tittelen på dokumentet. En ny implementering kunne forstørret hele Lupens tegnevindu i bakgrunnen, og bare forstørret vinduet 3 ganger som i dag.

7.4 En annen type nettkart: Visual Sitemap

Vi har sett på forskjellige typer nettkart hittil i oppgaven. De var alle sammen i likhet med Lupen basert på notasjonen om noder og lenker. **Forholdet mellom dokumentene** (lenkene) var svært sentralt. Vi trekker fram Visual Sitemap som et eksempel på et nettkart som har som primært formål å vise informasjon ut i fra et nettsteds **innhold**.



Figur 7-7 Visual Sitemap brukt på Smartmoneys hjemmeside

I Figur 7-7 ser vi et skjermbilde av Visual Sitemap fra SmartMoneys hjemmeside[37]. Innholdet er i fra børskursen i New York (Dow Jones). Kartet er delt opp i kategorier som gjenspeiler oppdelingen på børsen. Hver kategori blir delt inn i de største firmaene. Fargene varierer i intensitet fra tap (rødt) til gevinst (grønt). I figuren har vi valgt kategorien ”Health Care”, og firmaet ”Johnson + Johnson”. Vi ser at denne dagen falt firmaets børskurs med 1,18 %. Vi kan på menyen velge mye annen informasjon om de forskjellige firmaene.

Algoritmen som lager dette nettkartet er ukjent.

Denne layouten er svært interessant for nettsteder som har kursmateriale, som Opsys. Men kunne delt kartet opp i de forskjellige dokumentene/kapitlene/kategoriene for et kursmateriale.

[37] www.smartmoney.com/markedmap/. Eksemplet er ikke det beste, da ”innholdet” er børskursen fra Dow Jones i New York. Det var dessverre ikke mulig å finne andre skjermbilder, fordi de krevde registreringsavgifter.

Noder kunne forandret farge alt etter om de var besøkt eller ikke. Studentene kunne også selv gradert dokumentene i forhold til hvor godt de har forstått materialet (man måtte da laget en tjeneste for dette på hver enkelt side). Markering i rødt ville for eksempel indikert en dårlig forståelse, og at sider/kategorier måtte studeres mer.

Naturligvis ville ikke brukerne sett lenkene mellom de forskjellige nodene, men forfatter kunne også på dette kartet laget en ”sti”, som omhandlet forskjellige tema. Stien kunne blitt implementert som ett helt nytt skjermbilde (som ble valgt i en meny for eksempel). Hvis kartet hadde eksistert som et eget vindu, kunne navigeringsknapper gjort det mulig å velge retning, på samme måte som i Lupen.

7.5 Fremtiden til nettkartene

I en undersøkelse utført av Nielsen [Nie02] i 2001 hadde 48 % av alle undersøkte nettsteder statiske nettkart. Bare 27 % gikk til et nettkart når de ble bedt om å lære om nettstedets struktur. Når brukeren ble bedt om å gå til en side de nylig hadde besøkt på egenhånd, var bare 15 % klar over at det eksiterte et nettkart på siden.

Dette viser at nettkart på ingen måte er en vesentlig del av brukernes vaner på Internett. Man kan bare spekulere i årsakene, men det er nærliggende å tro at:

- Bruken av landemerker og bokmerker i nettleseren og i nettapplikasjonene, gir nok navigeringshjelp.
- Brukerne er nå vant med å benytte Internett, og de besøker mer og mer faste sider, istedenfor å ”surfe” rundt.

Microsoft prøvde å inkorporere dynamiske nettkart i Internet Explorer 4.0. Ideen var at nettkartet skulle være en del av nettleseren. Forfattere og Webmaster av et nettsted skulle lage automatisk eller for hånd, en indeksfil over innholdet. Denne filen skulle da nettkartet lese, og ut i fra den generere et oversiktsbilde over nettstedet. Hver node i nettkartet skulle inneholde følgende informasjon:

- Dokumentets tittel (skulle korrespondere til tittel i metadata)

- URL
- Eventuelle ikoner og deres URL
- Plassering i hierarkiet
- Filstørrelse
- Siste dato for editering
- Vis som node (true/false)

Av en eller annen grunn bestemte Microsoft seg for å forkaste disse planene. Det har ikke vært mulig å finne ut noe mer detaljert informasjon om dette, eller om planene blir gjenopptatt i fremtiden.

7.6 Internett er ennå ikke hypertekst

Et fremtidig dynamisk nettkart som fungerer på mer enn et nettsted ligger langt fram i tid:

- Internett følger for mange standarder og filformater.
- Dokumenter har ofte mangelfulle metadata. En edderkopp vil derfor ikke kunne kategorisere slike dokumenter på en effektiv måte.
- Det er ingen måte for et nettkart å hente ut lenkeklikk fra nettleserne. Dette gjør at nattkartet i så fall måtte være tilstandsløst.
- Metadata kan støtte mer konsise og relevante lenker mellom dokumenter. Disse er som regel ikke implementert i dokumentene.

Selv om man tenker WWW når man nevner hypermedia og hypertekst, er det en del ting som bryter med Vannevar Bush og Ted Nelsons ider. I deres imaginære modeller skulle alle typer noder og informasjon være støtte lenker. Modellen baserte seg på en fri flyt av tankegang (stier og lenker). Innhold skulle være konsistent, og statisk (Innholdet skulle ikke kunne slettes, og dermed bryte lenkene). I dagens arkitektur av WWW er det flere ting som gjør dette vanskelig og/eller umulig:

- Referanser i dokumenter er ikke alltid hyperlenker.

- Bruk av rammer. Man kan ikke direkte legge inn en lenke til et dokument, da rammer forutsetter at man går igjennom en prosess for å komme fram til dokumentet.
- Bruk av JavaScript/Java/CGI. Igjen må man gå igjennom en prosess for å finne informasjonen, istedenfor å lenke direkte.
- Registrering for å komme inn på en side, selv om man ikke skal betale for noe. Dette forhindrer direkte lenking, i alle fall for de som ikke har registrert seg.
- Bruk av <FORM METHOD = POST> i et CGI-script. Man kan ikke lenke til en utfylt versjon av FORM.
- Bruk av midlertidige nettsteder, e-postadresser og URL. Det er en høy risiko for at et nettsted/dokument ikke eksisterer etter en tid.
- Mangel på kreative lenker til andre nettsteder eller innad på sitt eget nettsted. Mange nettsteder bruker bare hypertekst til å presentere noen menyvalg, eller prøver å "fange" brukeren på sitt eget nettsted. Nielsen hevder det siste faktisk skremmer brukerne fra å besøke nettstedet flere ganger, såkalt "besøk en gang, så aldri mer". [Nie99]

Referanseliste

[And00] Andrews: How the web was won, 2000.

[Ber99] Bernard, Michael. Sitemap Design: Alphabetical or Categorical?, 1999. (http://psychology.wichita.edu/surl/usabilitynews/1s/sitemap.htm)

[Ber00] Bernard & Chaparro: Searching within Websites: A comparison of three types of sitemap menu structures, 2000.

[Bie97] Bieber et al. :Forth generation hypermedia: Some missing links for the World-Wide Web, 1997.

[Bie00] Bieber, Michael: “Hypertext," Encyclopedia of Computer Science (4th Edition), 2000.

[Bru99] Benjamin D. Brunk: Overview and Preview Tools for Navigation the World-Wide Web, 1999.

[Bus45] Vannevar Bush: As We may think, Atlantic Monthly 1945.

[Cle03] Clemens, Arthur: “the Diemen Patterns Repositoty”, (http://www.visiblearea.com/cgi-bin/twiki/view/Patterns/Site_map)

[Con87] Conklin, Jeff: Hypertext and introduction and survey, 1987.

[Con95] Conklin, Jeff: A survey of hypertext, 1995.

[Han93] Hannemann, Thuring & Haake: Hyper document Presentation: Facing the Interface, 1993.

[Hao99] Hao, Hsu, Dayal & Krug: Visual Mining Large Web-based hyperbolic space using hidden links, 1999.

[Ihl99] Carina Ihlstrøm:Navigation in large Web Sites, 1999.

[Kah01] Kahn & Lenk: Mapping Web sites, 2001.

[Low99] David Lowe & Wendy Hall: Hypermedia and the Web: An engineering approach, 1999.

[Mau96] Maurer, Hermann: Hyper-G, The second web solution, 1996.

[Mil98] Miller & Bharat: Sphinx: A Framework for creating personal, site-specific web crawlers, 1998.

[Muk] Sougata Mukherjea, James D. Foley et al, Visualizing the WWW with the Navigational View Builder.

[Nie90] Nielsen, Jakob: Hypertekst og Hypermedia, 1990.

[Nie99] Nielsen, Jakob: Designing Web Usability, 1999.

[Nie02] Nielsen, Jakob: ”Jakob Nielsens Alertbox”, (http://www.useit.com/alertbox/20020106.html)

[Rad91] Rada, Roy: Hypertext: From text to expertext, 1991.

[Ros] Rossi, Schwabe & Lyardet: Web application Models are more than conceptual models.

[Tau97] Tauscher & Greenberg: How people revisit web pages: empirical findings and implications for the design of history systems, 1997.

[Thu95] Thuring, Hannemann, & Haake: Hypermedia and Cognition: Designing for Comprehension, 1995.

[War00] Warholm, Hans Olav: Hypermedia I undervisningen, 2000.

[Zap] Panayiotis Zaphiris, Lianaeli Mtei: Depth vs. Breadth in the Arrangement of Web Links (http://www.otal.umd.edu/SHORE/bs04/index.html).

[Ros90] Rosengren, Peter: Hypermedia haller ratt på lapparna.

[Sch97] Schulmeister, Rolf: Hypermedia Learning Systems, 1997. (http://www.izhd.uni-hamburg.de/paginae/Book/default.html)

Litteraturliste (bakgrunnsmateriale)

Lewis & Loftus: Java Software Solutions, 1998.

Fan, Joel: Black Art of Java Game Programming, 1996.

Maurer, Herman: Hyper-G, The Second Generation Web Solution, 1996

Nielsen, Jakob: Designing Web Usability, 1999.

Kahn & Lenk: Mapping Web Sites, 2001.

Lowe & Hall: HyperMedia & the Web, an engineering approach, 1999.

Mendelzon, Alberto: Visualizing the World Wide Web, 1996

Plaisant, Grosjean & Bederson: SpaceTree: Supporting exploration in large node link tree, Design evolution and empirical evaluation.

Rossi, Schwabe & Guimaraes: Designing Personalized Web Applications, 2001.

Kreutz, Conradi & Spitzer: Improved Visual Navigation in Web-Documents, 1998.

Modjeska, David: Navigation in Electronic Worlds, 1997.

Ollerenshaw, Mark: Redefining the Browser History in Hypertext Terms, ????.

Leng Theng, Thimbleby & Jones: Reducing information overload: A comparative study of hypertext systems.

Weinreich, Obendorf & Lamersdorf: The Look of the Link - Concepts for the User Interface of Extended Hyperlinks.

Berners-Lee, Tim: Information Management: A proposal (www.w3.org), 1990.

Lamping, Rao & Pirolli: A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies, 1995.

Tunkelang, David: A Numerical Optimization Approach to General Graph Drawing, 1999.

Mukherjea, Foley & Hudson: Visualizing Complex Hypermedia Networks through Multiple Hierarchical Views.

Eilertsen & Mikalsen: Linux tjenestedrift, 2003.

Warholm, Hans Olav: Hypermedia I undervisningen, 2000.

Relevante lenker

Opsys	mserver.idi.ntnu.no
The Brain	www.thebrain.com/Default.htm
Sun Java Forum	forum.java.sun.com/
WebSPHINX	www-2.cs.cmu.edu/~rcm/websphinx/#download
Xanadu	www.xanadu.com.au
INXIGHT	www.inxight.com/map/
HyperWave	www.iicm.edu/hgbook
The Java Tutorial	java.sun.com/docs/books/tutorial/?frontpage-spotlight
LiveConnect	www.idi.ntnu.no/manualer/ServerSideJS/lc.htm
JavaScript	devedge.netscape.com/central/javascript/
Perl tutorial	wwwacs.gantep.edu.tr/programming/perl-ebook/
Apache http project	httpd.apache.org/docs-project/
Jbuilder	www.borland.com/jbuilder/
Innsida	www.innsida.ntnu.no